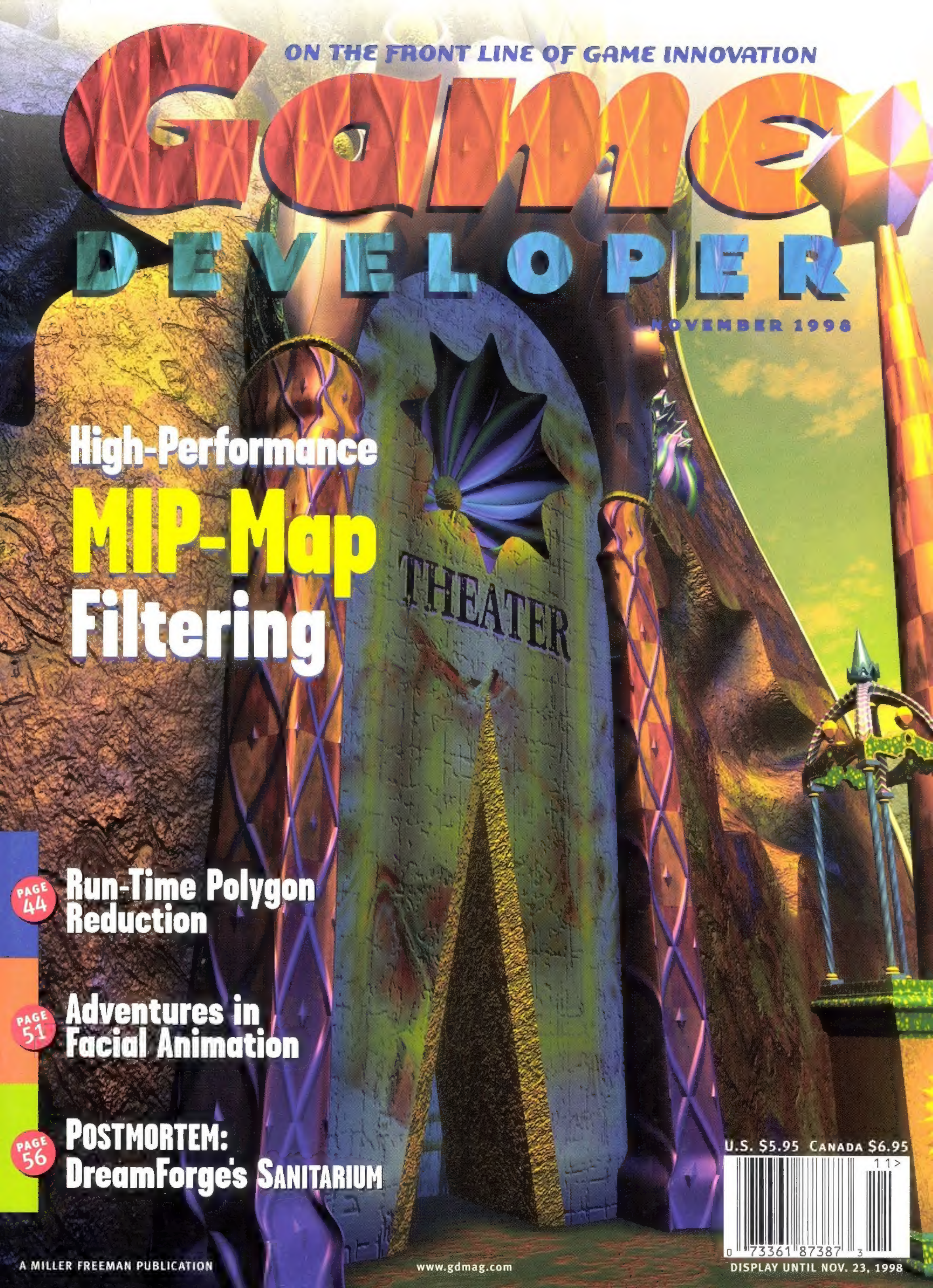
ON THE FRONT LINE OF GAME INNOVATION

Game Developer

NOVEMBER 1998



High-Performance MIP-Map Filtering

PAGE 44 **Run-Time Polygon Reduction**

PAGE 51 **Adventures in Facial Animation**

PAGE 56 **POSTMORTEM: DreamForge's SANITARIUM**

U.S. $5.95 CANADA $6.95

11>

0 73361 87387 3

A MILLER FREEMAN PUBLICATION

www.gdmag.com

DISPLAY UNTIL NOV. 23, 1998

The Art of War

IN GAME DEVELOPMENT, TIME AND MEDIOCRITY ARE THE ENEMY. IT'S NEVER ENOUGH TO JUST FINISH ON TIME AND ON BUDGET, YOUR GAME MUST BE ADDICTIVE AND FUN TO PLAY.

SOFTIMAGE®|3D offers game-specific tools for polygonal modeling, character animation sequencing, and color space manipulation. The new SOFTIMAGE®|GDK (Games Development Kit) allows for quick and easy import/export tool creation. Full development environments are available for SEGA Saturn, Nintendo 64, SONY PlayStation, and DirectX platforms.

Conquer your game faster. Don't limit your creativity.
MAKE CONTACT:

SOFTIMAGE®|3D
softimage.com/game_dev
1 800 576-3846 x8805 1 514 845-1636 x8805

Credits from left to right • Image from Riven® © 1997 Cyan, Inc.
© 1996 Nintendo. Images courtesy of Nintendo of America Inc.
Hexen® II 1996 by Raven Software, Inc.

© Copyright 1998 Avid Technology, Inc. All rights reserved.

Virtua Fighter 3 • © SEGA Enterprises, Ltd.

CONTENTS

VOLUME 5, NUMBER 11

NOVEMBER 1998

FEATURES

34 Run-Time MIP-Map Filtering

Texture maps can suffer from several unfavorable visual effects. This math-heavy but computationally efficient method of generating MIP-maps can filter and scale textures for sharp-looking texture maps at varying LODs.

BY ANDREW FLAVELL

44 A Simple, Fast, and Effective Polygon Reduction Algorithm

If you're looking for a way to reduce the complexity of your 3D models on the fly, here's a fast and elegant algorithm with a number of real-time applications.

BY STAN MELAX

51 Animating Facial Expressions

Facial motion capture can either add realism to your game or really give your players the creeps. If you're interested in the former rather than the latter, Rodgers has some common-sense guidelines for you.

BY JAKE RODGERS

56 Postmortem: DreamForge's SANITARIUM

A strong and inspired design coupled with some innovative team management guidelines and solid asset management tools made this development effort smooth and sane.

BY CHRIS PASETTO

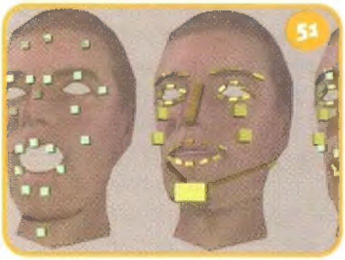
COLUMNS

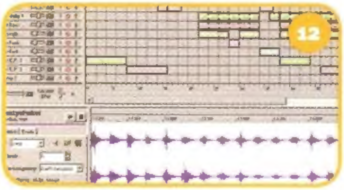
15 **Graphic Content** BY JEFF LANDER
Making Kine More Flexible

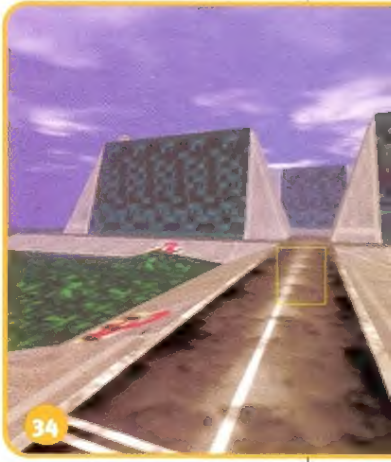
25 **Artist's View** BY MEL GUYMON
Painting With Vertex Colors

31 **Hard Targets** BY OMID RAHAMT
Trends in the Sales Channel

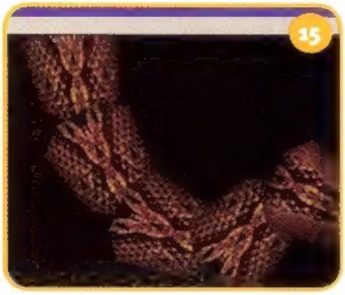
72 **Soapbox** BY JEANNE COLLINS
Turbo-Charge Your Test Lab

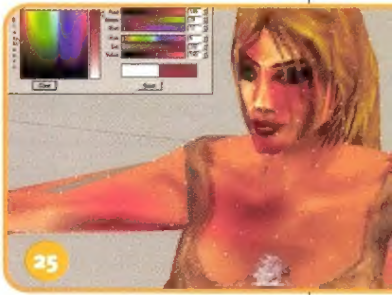

51


12


34

56


15


25

DEPARTMENTS

2 **Game Plan** BY ALEX DUNNE

4 **Says You!** LETTERS FROM OUR READERS

6 **Bit Blasts**
The latest new tools for game development, industry happenings, and our new capsule product review section.

COVER: *Cover art created by Gil Bruvel, on a UNIX workstation, for the CD-ROM game* OF LIGHT AND DARKNESS, *which was produced by Interplay Productions. You can see more Bruvel art work at http://www.bruvel.com.*

Game Developer (ISSN 1073-922X) is published monthly by Miller Freeman Inc., 600 Harrison St., San Francisco, CA 94107, (415) 905-2200. Please direct advertising and editorial inquiries to this address. Canadian GST: 124513185. **SUBSCRIPTION RATE** Subscription rate for the U.S. is $34.95 for twelve issues. Countries outside the U.S. must be prepaid in U.S. funds drawn on a U.S. bank or via credit card. Canada/Mexico: $44.95; all other countries: $54.95 (issues shipped via air delivery). Periodical postage paid at San Francisco, CA and additional mailing offices. **POSTMASTER:** Send address changes to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. **Customer Service:** For subscription orders and changes of address, call toll-free in the U.S. (800) 250-2429 or fax (847) 647-5972. All other countries call (1) (847) 647-5928 or fax (1) (847) 647-5972. Send payments to *Game Developer*, P.O. Box 1274, Skokie, IL 60076-8274. For back issues write to *Game Developer*, 1601 W. 23rd St. Ste. 200, Lawrence KS 66046-2703. Call toll-free in the U.S./Canada (800) 444-4881 or fax (913) 841-2624. All other countries call (1) (913) 841-1631 or fax (1) (913) 841-2624. Back issues are $7.00 each postage-paid ($5 each for quantities of five or more) for U.S. orders and $9.00 each postage-paid ($7 each for quantities of five or more) for international orders. Please remember to indicate *Game Developer* on any correspondence.

The Second-Title Trap

Game development is no walk in the park, especially when it comes to managing the business aspects. And possibly the most stressful period for a studio is when it finds itself falling behind schedule.

Missing a milestone has several ramifications. The two most notable side effects are a souring of the developer-publisher relationship and the postponement of milestone payments. The latter is especially serious for developers, and can propel a studio into crisis mode. Creditors go unpaid, medium- and long-term planning go out the window, and the team pulls together to satisfy immediate goals.

If these consequences weren't bad enough, often it gets worse. Pressures can lure developers into the "second-title trap." Faced with a high burn rate and limited funds to complete a title, it's common for a development studio to pull some people off of the current project to put together a design document for a second title, in the hope that after a couple of months and more contract negotiations, a publisher will green-light it. The second revenue stream would offset the lagging milestone money from the first project, buoying the company for a while.

Watch out. Don't forget that many publishers insist on the first right of refusal for a developer's second title, so the developer has to approach his current publisher first. If the first project is going poorly, this publisher probably knows it, and is therefore bargaining from a position of strength. If the publisher gives the second project the thumbs-up, this contract will probably be much less favorable for the developer than the previous one.

If the publisher punts on the second project, the developer may decide to look for funding elsewhere. So now the developer, who hasn't been talking up other publishers in a while, must shop the second project around the industry. Back in the office, the original game may fall further behind because key staffers are absent. Bills for plane trips and hotel rooms from the far-flung schmoozing trips pile up in the office. And here we go down the spiral. Wheee!

If you find yourself in this situation, my condolences. Your management skills will be put to the test, and hard decisions regarding layoffs, budgets, and vacation plans will be forced upon you. One tidbit of advice: if your publisher is ranting at you over a blown deadline, maintain your cool. It's an emotional time, and taking the high road rather than getting into a war of words will help preserve your relationship.

Of course, there are ways to avoid falling into this morass at all. Account for downtime in your budget. When figuring out how much money you need to complete a title, factor in a couple of months' worth of operational capital to get by after the product is completed. Don't tap into that money if you run short. Instead, immediately go back to your publisher, confess that you've gone over budget, and face the music. Running out of money before you deliver the title puts you in a better position than if you run out after you've delivered the completed product to your publisher. The undelivered product is your only leverage during development.

Never factor royalty payments into your business plan. Think of royalties as bonuses or stock dividends (windfall). If your budget depends on royalty payments from your games, rest assured that you'll run out of money.

Don't wait until the last second to put together a design document for the second title. Presenting a publisher with a second proposal earlier forces the publisher to either fund it or fan it more quickly. If it decides to pass, you can pursue other publishers before the cash from the first project dries up. If possible, plan two titles from the start, using two different publishers. Of course you'll want to stagger the delivery dates.

Finally, brush up on your business management skills. I highly recommend Gordon Bell's *High-Tech Ventures: The Guide for Entrepreneurial Success* (Addison Wesley, 1991). Bell's been around the block many times, and the case studies in his book offer priceless lessons. ■

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

600 Harrison Street, San Francisco, CA 94107
t: 415.905.2200 f: 415.905.4962 w: www.gdmag.com

Publisher
Cynthia A. Blair *cblair@mfi.com*

EDITORIAL

Editor-in-Chief
Alex Dunne *adunne@sirius.com*
Managing Editor
Tor D. Berg *tdberg@sirius.com*
Departments Editor
Wesley Hall *whall@sirius.com*
Art Director
Laura Pool *lpool@mfi.com*
Editor-At-Large
Chris Hecker *checker@d6.com*
Contributing Editors
Jeff Lander *jeffl@darwin3d.com*
Mel Guymon *mel@surreal.com*
Omid Rahmat *omid@compuserve.com*
Advisory Board
Hal Barwood LucasArts
Noah Falstein The Inspiracy
Brian Hook id Software
Susan Lee-Merrow Lucas Learning
Mark Miller Harmonix
Paul Steed id Software
Dan Teven Teven Consulting
Rob Wyatt DreamWorks Interactive

ADVERTISING SALES

Western Regional Sales Manager
Alicia Langer *alanger@mfi.com* t: 415.905.2156
Eastern Regional Sales Manager
Kim Love *klove@mfi.com* t: 415.905.2175
Sales Associate/Recruitment
Ayrien Houchin *ahouchin@mfi.com* t: 415.905.2788

ADVERTISING PRODUCTION

Vice President Production Andrew A. Mickus
Advertising Production Coordinator Dave Perrotti
Reprints Stella Valdez t: 916.983.6971

MILLER FREEMAN GAME GROUP MARKETING

Group Marketing Manager Gabe Zichermann
MarComm Manager Susan McDonald
Marketing Coordinator Izora Garcia de Lillard

CIRCULATION

Vice President Cirulation Jerry M. Okabe
Assistant Circulation Director Mike Poplardo
Circulation Manager Stephanie Blake
Circulation Assistant Kausha Jackson-Crain
Newsstand Analyst Joyce Gorsuch

INTERNATIONAL LICENSING INFORMATION

Robert J. Abramson and Associates Inc.
President Libby Abramson
720 Post Road, Scarsdale, New York 10583
t: 914.723.4700 f: 914.723.4722
e: *abramson@prodigy.com*

Miller Freeman
A United News & Media publication

Chairman-Miller Freeman Inc. | Marshall W. Freeman
President/COO | Donald A. Pazour
Senior Vice President/CFO | Warren "Andy" Ambrose
Senior Vice Presidents | H. Ted Bahr, Darrell Denny Galen A. Poss, Wini D. Ragus, Regina Starr Ridley, Andrew A. Mickus, Jerry M. Okabe
Vice President/SD Show Group | KoAnn Vikören
Senior Vice President/Systems and Software Division | Regina Ridley

BPA International Membership
Applied for March 1998

www.gdmag.com

DEMONSTRATION:

1. Tear out this page.
2. Crumple into ball.
3. Toss over your shoulder.

▶ The sound this ad will make when it hits the ground *behind* you is the power of A3D™ from Aureal, the only audio that works the way your ears do.

No wonder game makers like Interplay and GT Interactive are featuring A3D in their newest titles. Experience it for yourself at www.A3D.com

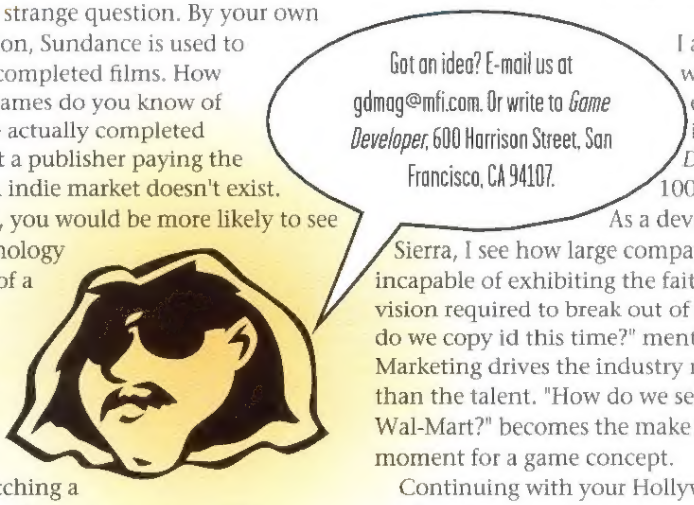
A
3D

BELIEVE YOUR EARS™

©1998 Aureal Semiconductor, Inc. A3D and Aureal are trademarks of Aureal Semiconductor Inc. Other trademarks are properties of their respective owners.

Indie Game Festival? Take Two.

Where's our Sundance? (from September 1998 GamePlan) What a strange question. By your own admission, Sundance is used to screen completed films. How many games do you know of that are actually completed without a publisher paying the bill? An indie market doesn't exist. Instead, you would be more likely to see a "technology demo" of a game which may "wow" — but this is like watching a trailer instead of a completed film.


Got an idea? E-mail us at gdmag@mfi.com. Or write to *Game Developer*, 600 Harrison Street, San Francisco, CA 94107.

Indie films are financed out of pocket, in the hope that the producers will gain big when a distributor picks it up. There are over 80 years of procedure and analysis for how a film producer will make his money back, including the fact that technology for showing the media already exists in mass formats. Every time a movie shows, the producer gets a check.

Currently, the best an "Independent Game Producer" could see is his money back (ooh) and perhaps a small royalty (the bulk of it goes to the game developers) if the game's sale recoups its cost. If the game fails, that's it, it's gone forever (no market for residuals, as old games are low tech.). Being an independent game producer would be great for tax write-offs, but not much for income.

Indie films are also quickie films. The production is complete in under a month. Games can take 6 to 12 production months (especially if low budget). That's a lot of time to have one's money tied up with no guarantees. Bottom line, this business needs to mature and become more profitable. Then, and not before, we can have comparisons with Sundance (or the Oscars or Variety, or many other Hollywood examples we do not currently fit).

Now the obvious. E3, as you noted, is not the Sundance equivalent. But the Computer Game Developers Conference [now the GDC] is as close as you are going to get today. Networking, seminars, demos and yes, even a completed independent game may be found there (in a relaxed venue).

Tony Van
Producer, Electronic Arts

I agree with your editorial in *Game Developer* 100 percent. As a developer for Sierra, I see how large companies are incapable of exhibiting the faith and vision required to break out of the "How do we copy id this time?" mentality. Marketing drives the industry more than the talent. "How do we sell this to Wal-Mart?" becomes the make or break moment for a game concept.

Continuing with your Hollywood analogy, too many games are put into production on pitches such as, "It's a cross between Quake and Diablo with a little Myst thrown in." OpenGL, DirectX, and the Internet have provided indie developers with all of the resources needed to make a great game but little chance to get that game to market.

An added benefit, if this could grow to other cities, would be the contact between gameplayers and developers. At screenings, indie filmmakers get instant feedback on what worked and what didn't allowing them to improve on the next one. Game players also get insight into why decisions are made in the games they like or don't like. E3 does not provide that kind of contact.

Jim Edwards
via e-mail

Your idea promoting the organization of an independent game developer festival is intriguing, but given the nature of the software development industry, I'd be surprised to see anything like it take shape. E3, obviously, is not a festival so much as it is a conference — a place where established companies showcase their products before other industry executives, and where even smaller development houses typically have the support of a larger parent company. To bring a game concept to playable demo level, much less completion, requires a lot of resources (money, talent, equipment, salaries, and so on) that don't exist in the same way that they do in the film industry. To top it off, smaller companies developing a product for, say, a popular console, must do so in the usual entanglement of legal secrecies and NDAs, as well as laboring under the financial responsibilities of licenses, development kit purchases, and other complications.

Maybe the landscape for independent development would be different if things like the black PlayStation (Yaroze) were more widespread. The problem is, once you're involved in technology, you're involved in licensing. Software products don't stand alone as autonomous entities the way films do (meaning: platform and hardware, not marketing and distributing).

There are so many other problems — the time to develop a game vs. the time to shoot a film (16 weeks for a film compared to over a year for a game), the presence of independent sources of money for filmmaking, and so on. The notion that a team of developers can bring a product to bear, showcase it, only to then have it picked up and distributed is, frankly, almost absurd.

There are currently a number of festivals which highlight the underground of digital filmmaking (D.FILM, ResFest), and perhaps something that provides a showcase for all independent digital creativity (games, films, animation, graphic design, and the like) would be more likely to succeed.

As a video maker and game animator myself, I would be more interested in a project that is more encompassing. Let's not forget that Sundance has become an inflated feeding frenzy, criticized profusely and abandoned by many members of the independent film community.

Sean Capone
via e-mail

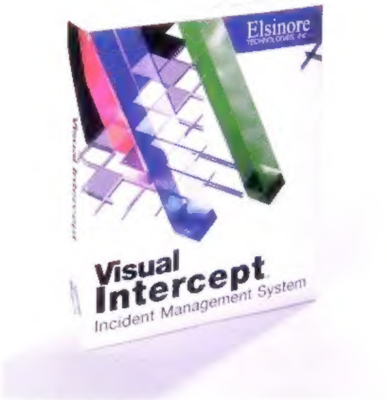
Due Credit for the October Cover Image.

The Skaar from GT Interactive/Epic MegaGame's Unreal, was modeled by Dave "Motornerve" Carter. Dave built this image with Kinetix's 3D Studio R4, BonesPro from Digimation, and proprietary texture-mapping tools. It should be noted that the image shows actual in-game models, skins, and environments. Dave can be reached at motonerv@xnet.com.

Visual Intercept™

The ONLY bug-tracking system integrated with Microsoft Visual Studio.™

Microsoft
Microsoft Visual Studio 97
Enterprise Edition



Elsinore Technologies, Inc.
Visual Intercept
Incident Management System

Suite.

Sweet.

The ONLY fully-integrated, project-oriented enterprise bug tracking system for Microsoft Visual Tool Users.

- Software Developer's Kit for Building Customized Solutions
- Open Architecture: 3-Tier, ODBC, 32-Bit
- Scalability: Support Hundreds of Users on the Same System
- Built-in Reporting and Automatic Notification

Now WEB-enabled!

FREE DOWNLOAD!
www.elsitech.com/vid

Call now! 1.800.364.5467 RGI

Elsinore
TECHNOLOGIES, INC.™
The Leader in Incident Management

Visual Intercept™
Incident Management System

Microsoft Tool Users... We Know Where You Live.

BIT Blasts

News from the World of Game Development

New Products: RTIME Interactive Networking Engine V3.0, 3Dfx's Glide 3.0, and Okino's new version of PolyTrans **p. 6**

Industry Watch: Many mergers, ups and downs at 3Dfx, and Broderbund's layoffs **p. 8**

Product Reviews: Intel's VTune 3.0 and Sonic Foundry's Acid **p. 10**

New Products

by Wesley Hall

New Network Engine

RTIME recently released the latest version (V3.0) of the RTIME Interactive Networking Engine, a scalable, high-performance client/server networking engine that supports real-time, multiplayer gaming over both the Internet and Local Area Networks (LANs).

The engine enables game developers and publishers to build or port multiplayer games to the Internet. This new release is built on top of a completely rebuilt core engine, extends the engine's performance and capabilities, and includes new features in response to the requests of RTIME's customers. One such feature is the new RTIME Integrated Server. Now RTIME-enabled games no longer require a dedicated server to act as a host. A client application can now link directly with a local server library that services all other connected gamers. V3.0 also helps reduce bandwidth consumption through the use of server-side dynamic filter management. The Parallel WorldMasters feature extends the previous WorldMaster (a server-side, super-client used to drive inanimate objects, provide server-based authority and security, maintain scores, and so on) to enable multiple copies to run in parallel. Each one can be used to control different simulations in the virtual world. Other new features include the ability to specify restriction information on public and private objects, enhanced transport directives, and streamlined object-level API. These build on already existing functions such as the Distributed Timebase Manager (to keep global time synchronization), the Realtime Filter Manager (uses affinity-based data distribution to determine the appropriate information to send each player in realtime), the Intelligent Update Manager, and Server-to-Server Communications. These features are beneficial for many game genres, including action, strategy, sims, sports titles, and persistent worlds. RTIME V3.0 is currently being integrated into several fall titles, including Acclaim's TUROK 2: SEEDS OF EVIL, and Ripcord's SPECOPS: RANGERS LEAD THE WAY.

The RTIME Client runs on Windows 95, Windows NT, SGI, and Solaris. The RTIME Server runs on SGI, Solaris, and Windows NT. A development-only version is available for Windows 95.

■ **RTIME Inc.**
Seattle, Wash.
(206) 281-7990
http://www.rtimeinc.com

Glide 3.0

3DFX just announced Glide 3.0, the new version of its low-level software interface that enables control of the company's Voodoo family of graphics chips.

Glide can serve as the primary application program interface (API) or may work in conjunction with another API to enable optimal acceleration on 3Dfx hardware. The API is designed specifically and only for 3Dfx hardware. Glide 3.0 is more streamlined than previous versions in an effort to make it easier to write complex games or applications that can fully take advantage of present and next-generation 3Dfx chips. Glide 3.0 utilizes triangle strips and fans, and has new features including gamma table support for complete control over lighting brightness (including individual color brightness control), vertex layout support for complete control over individual vertices, and an extension mechanism to allow developers to add additional functionality and optimizations.

The Glide software developer's kit (SDK) is available free of charge via the 3Dfx web site. The kit also includes Glide programming libraries, tools, and sample code. Glide 3.0 and its SDK are available for download immediately.

■ **3Dfx Interactive Inc.**
San Jose, Calif.
(408) 935-4366
http://www.3dfx.com

Updated PolyTrans

OKINO recently updated its stand-alone PolyTrans Model Translator (for Windows and SGI) to version 2.2.

PolyTrans is a model translation program that allows 3D models and scene files to be converted between various industry standard file formats in their entirety. Rather than convert just simple geometry and some material attributes, the PolyTrans program converts every aspect of a file so that the translated file is "render ready." New features include: animation conversion, auto-bitmap conversion for RIB exporter; animation conversion between 3D Studio MAX, Lightwave, RIB, and DirectX; Direct plug-in support for 3D Studio MAX v1.2, v2.0, and v2.5; support for several new file formats; and fully integrated trimmed NURBS support.

PolyTrans is available for Windows 95/NT and SGI. Pricing starts at $395 (PC, Windows) and $495 (UNIX).

■ **Okino Computer Graphics Inc.**
Mississauga, Ontario
(905) 672-9328
http://www.okino.com

IT'S RENDER HAPPY. Imagine your apps with cartoon rockets strapped to their feet. That's how they run on the new IBM IntelliStation.™ Its two screaming microprocessors make applications from vendors like Adobe, Avid and Alias|Wavefront literally scream with delight. What's that mean to you? Less time waiting. More time creating. Oh yeah, and happy killer graphics. Visit www.ibm.com/intellistation or call **1 800 IBM 7255, ext. 5007.** Careful, though. What you find could render you speechless.

THE IBM INTELLISTATION WORKSTATION.™ Windows NT® / Up to two Intel Pentium® II Xeon™ processors (450 MHz) / Intel 440GX AGPset with 100 MHz front-side bus / Up to 2GB SDRAM ECC memory / Matrox Millennium II / G200 / Permedia 2A / Intergraph Intense 3D Pro 3400 / Starting at $2200*

e-business tools

IBM®

IntelliStation
IBM
M Pro
IBM

intel inside
pentium® II
xeon™
PROCESSOR

*Estimated reseller price to end users for model 6893-10U. Actual reseller prices may vary. Monitor sold separately. MHz refers to microprocessor speed. Certain factors will affect application performance. PCs referenced in this ad include an operating system. Other company, product or service names may be trademarks or service marks of others. IBM product names are trademarks of International Business Machines Corporation. Intel Inside and Pentium are registered trademarks and Pentium II Xeon is a trademark of Intel Corporation. ©1998 IBM Corp. All rights reserved.

Industry Watch

by Alex Dunne

EIDOS announced record results for the three months ending June 30. It saw record revenues to the tune of £25.8 million (up from £9.4 million last year), amounting to a net loss of £3.0 million (compared to a £6.8 million loss last year). In other news, the company acquired Crystal Dynamics in a deal valued at about $47.5 million, but don't expect any Gex-Croft collaborations.

GT BAGS FUGITIVE. GT Interactive acquired equity interest in Fugitive Studios, a recent startup cofounded by Greg Williams, James Phinney, Jess McReynolds, and Brian Sousa. Many of the Fugitive employees were among the contingent that bolted from Blizzard after StarCraft shipped. In the deal, GT gets exclusive global publishing rights to Fugitive software titles for both PC and console platforms, plus print and merchandising rights. Fugitive's first title will be an "innovative 3D game" for the PC, and should ship in late 1999.

AS EXPECTED, Broderbund layoffs will be heavy as a result of the company's acquisition by The Learning Company. Approximately 500 of Broderbund's 1200 jobs will be trimmed. Half of these cuts will come from Broderbund's California operations in Petaluma (which is closing this month) and Novato. The company hasn't yet decided where the other 250 layoffs will come from.

LUCASARTS SIGNED an exclusive two-year publishing and distribution agreement with Activision, whereby Activision will handle all upcoming LA titles in the United Kingdom, Scandinavia, Central Europe, the Middle East and certain African countries.

ELECTRONIC ARTS completed its previously announced acquisition of Westwood Studios from Virgin Interactive Entertainment (which is itself a division of Spelling Entertainment Group). As a result of the $122.5 million cash deal, Westwood Studios becomes EA's tenth studio. Brett Sperry and Louis Castle have agreed to five-year employment contracts, and will remain with Westwood.

DISCREET MAX! The 3D tools industry has seen quite a bit of activity lately, and plots continue to unfold. Following closely on the heels of Microsoft's sell-off of Softimage to Avid, Autodesk acquired Discreet Logic for $520 million, creating quite a powerhouse of digital content creation tools. As a result, Autodesk's Kinetix division has merged into the new Discreet division, and tools such as 3D Studio MAX will now carry the Discreet name.

NEW PARADIGM. In the world of real-time 3D tools, MultiGen and Paradigm Simulation merged, creating MultiGen-Paradigm Inc. MultiGen, with its strong modeling tool Creator (recently highlighted in the August 1998 Postmortem of Atari's San Francisco Rush) is a good match-up with Paradigm's simulation tools, such as Vega. MultiGen will continue its operations in San Jose, and Paradigm offices will remain in Dallas, and the two firms will combine their worldwide sales operations.

3DFX'S MIXED NEWS. The burgeoning 3D hardware market isn't all happiness and joy. In fact, it's getting awfully crowded on those store shelves. 3Dfx announced that its Q3 retail sales figures would be lower than anticipated, due a slowdown in the retail channel. Now there's a glut of 3D inventory in the hands of retailers, and the company is pinning hopes on a healthy Christmas season to clear out some inventory. Assuming there's no increase in demand, 3Dfx anticipates losing several million dollars at the pre-tax operating level for its third fiscal quarter. Fortunately for the firm, its recent settlement with Sega (over the Dreamcast deal) gave it some wiggle room, and 3Dfx still anticipates a profitable third quarter. Following 3Dfx's announcement, Robertson Stephens downgraded 3Dfx shares from "Strong Buy" to "Buy", and the stock at press time is trading at its 52-week low (about $9).

On the upside for 3Dfx, software retailers Babbage's and Software Etc. just announced the creation of a special 3Dfx section within their 450 nationwide stores. This 3Dfx-only area will sell 3Dfx-related hardware and software products, such as the Diamond Monster Fusion and Creative Labs' 3D Blaster Voodoo2, plus 3Dfx-optimized titles such as Unreal, NFL GameDay '99, Need For Speed 3: Hot Pursuit, and Final Fantasy VII. ■

UPCOMING EVENTS

Game Developers Conference RoadTrip: Seattle

WASHINGTON STATE CONVENTION AND TRADE CENTER
Seattle, Wash.
November 9-10, 1998
$225
www.gdconf.com/1998/road-trips

Game Developers Conference RoadTrip: Austin

AUSTIN CONVENTION CENTER
Austin, Texas
November 16-17, 1998
$225
www.gdconf.com/1998/road-trips

Game Developers Conference RoadTrip: S. San Francisco

SOUTH SAN FRANCISCO CONFERENCE CENTER
South San Francisco, Calif.
November 21-22, 1998
$225
www.gdconf.com/1998/road-trips

Digital Content Creation

LOS ANGELES CONVENTION CENTER
Los Angeles, Calif.
December 2-4, 1998
$595
www.dccexpo.com

MicroScribe-3D™

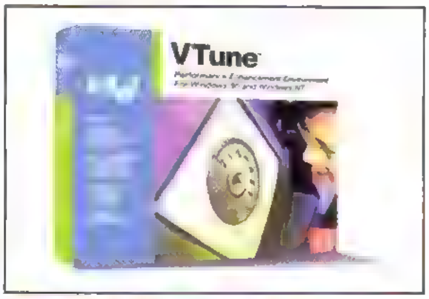
Desktop Digitizing Systems

MicroScribe 3D is an advanced 3D digitizing tool. Microscribe empowers you with the ability to capture complex 3D models from physical objects by tracing over their surfaces. The 3D digitizing process is fast, accurate, and more affordable than you might think!

The MicroScribe 3D is a desktop system that easily connects to your serial port and runs on PC, Mac, and SGI platforms. MicroScribe works directly in your favorite 3D graphics and CAD packages such as 3D Studio Max, Alias, Lightwave, AutoCAD, Solidworks, form Z, Rhino 3D, Amapi Studio, and many more. MicroScribe comes in several size and accuracy configurations to meet the demands of your project.

For details, call Immersion at 408-467-1900 or 1-800-893-1160 or visit our web site at www.microscribe.com.

Immersion
Corporation



VTune 3.0

by Dan Teven

If it seems as though we've devoted a lot of coverage to Intel's VTune profiler this year, you're not imagining it. Ron Fosner awarded four stars to VTune 2.5 in an April review. Then we wrote up VTune again, in June, when it won the Front Line Award for Programming Utilities. And now, despite *déja VTune*, I'm going to tell you why release 3.0 is worth an additional half star.

I'll assume you've seen an earlier incarnation of the product, or at least read Fosner's review (which is online at www.gamasutra.com/tools/reviews/). While VTune has always been a great product for optimization junkies, its old user interface was seriously flawed — Fosner called it "cluttered" and "annoying." Happily, Intel has jettisoned the old UI and replaced it with a far superior one.

Releases of VTune have always advanced the art of performance measurement, and 3.0 is no exception. You can now view an annotated function-call hierarchy, showing explicitly how many times a function was called as well as the time taken by the function and its descendants. You can also correlate operating system events such as context switches or page swaps with your application's behavior. The new information dovetails nicely with conventional hotspot analysis, resulting in a more complete picture of performance.

QUICK START. The first time you run VTune, you'll see an Easy Start menu with three options: Quick Performance Analysis, New Project Wizard, and Open VTune Project. All of these tasks are just as easily accomplished from the File menu, and you have the option of turning off Easy Start.

Quick Performance Analysis samples the current CS:EIP of each processor — yes, VTune 3.0 supports multiprocessor systems — every millisecond for 20 seconds. It automatically charts the results by process, by processor, and by module, in three tabs of a single window. A second window shows the hotspots within the main module you've specified. The main module doesn't have to be an executable file; it can be a .DLL, an .OCX, a device driver, a Java class, or even an object module.

Quick Performance Analysis is an excellent place to begin when you're profiling anything that takes less than 20 seconds. The two-window view is just right. You not only find the bottlenecks in your main module, you find out if other modules are consuming more time than you expected.

Behind the Easy Start menu, the old postage-stamp-sized main window (with multiple pop-ups) has been superseded by a full-size main window (with multiple children). Child windows can be tiled, minimized, and maximized. Not only is this layout more familiar, it's less cluttered.

The Navigator window, down the left side of the screen, is a tree control that lets you switch quickly among myriad sampling sessions and views of the data. I like this feature, but I do some of my work on a notebook with a 640×480 display, and it's not worth the screen real estate in that situation.

DRILLING DOWN. More improvements are evident as soon as you try to drill down to the instruction level. In earlier versions, you would start with a system-wide view and select the module that interested you. This would pop up a new window showing hotspots in that module. Then you'd select the hotspot that interested you — and pop up another window. Eventually, you'd get to a window with a source or disassembly view. To make matters worse, you'd frequently have to perform these operations by selecting a pixel-wide bar with your mouse, and VTune's hit testing was sometimes off by a couple of pixels!

Now, VTune opens the hotspot window for the main module automatically, saving a step. And you don't get a new window every time you zoom in; instead, windows are recycled. It's still possible for the hotspots to be a pixel or two wide, and I wish Intel would implement a magnifier tool to make the selection process easier, but the mouse click precision bug is fixed.

Whether you're looking at graphs or raw sample data, time-based or event-based sampling, or static or dynamic code analysis, all windows share the same main menu and toolbar. This is a big improvement. In earlier releases, where windows had their own individual interfaces, I often had trouble figuring out how to get from point A to point B.

The source and disassembly views haven't changed much since earlier releases. It's easy to see where you're spending CPU cycles, and VTune does an excellent job of explaining pairing issues and execution penalties. I was happy to find the reference manuals for the Pentium Pro instruction set and for Intel's MMX intrinsic functions added to the online help. Likewise, the event-based sampling feature hasn't changed much.

CALL GRAPH ANALYSIS. Consider a program that decompresses a bitmap and then, accidentally, copies it to the screen twice. Hotspot analysis by itself won't reveal the problem. Say you profile with VTune 2.5 and observe that half the time is spent in your decompression code, with the rest spread out among the operating system (`BitBlt`) and video drivers. You optimize the decompressor, but the program's still too slow, and you're stymied.

Fortunately, VTune 3.0 lets you do call graph profiling. This means collect-

Not so long ago, Dan Teven was obsessed with making a really cool game go really, really fast. This morning, he saw the game in a bargain bin for $15. Write him at dteven@ici.net to commiserate.

ing data about every function call in your program: not only who called whom, but how many times, and how much time elapsed between the function call and the return. You find that a decompression call takes twice as long as a call to `BitBlt`; from the hotspot analysis, you'd have expected the calls to take equally long. This can only mean you're calling `BitBlt` too often. VTune reveals the extra call, from an unexpected place. Bingo — another 25 percent speed improvement.

I'm a firm believer in using call-timing data to crosscheck hotspot data. In fact, I've often instrumented my projects with function timing calls. Now that I can use VTune to gather the same information — more thoroughly and with better viewing capabilities — I'll be retiring that code.

The only downside to call graph profiling is that it affects the performance of a program. The instrumented calls are slower to execute, and the extra code can have adverse effects on the cache. VTune gives you some basic choices about which modules to instrument, but I'd really like to see an API so I could have fine-grained control.

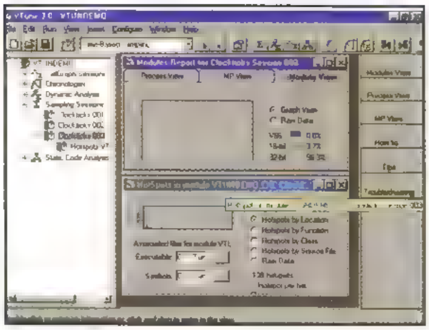
CODE COACHES. VTune now has code coaches for five languages: C, C++, Java, Fortran, and assembly. The coaches are surprisingly skilled. I ran the C coach on a program with an obvious bottleneck: a brute-force check for duplicate strings in a list. The bottleneck is so obvious that a comment in the source code warns about it and suggests binary searching as a fix. VTune not only recognized that the problem lay with the algorithm, it offered better advice than the comment, suggesting both binary searching and hashing as solutions.

The assembly coach, which is new in 3.0, looks for assembly-level optimizations such as instruction scheduling and partial stall elimination. It won't find much that a good optimizing compiler wouldn't, but it would be an excellent learning tool for an assembly-language programmer.

EVENT CHRONOLOGIES. Two new features that promise more than they deliver are Processor and OS Chronologies. Essentially, VTune tracks various events during the session — processor events, such as cache misses, or OS events, such as page swaps (the same ones you can display in System Monitor) — and graphs the data over time. Selecting a time range on the graph opens a new window illustrating the modules that were active during that time period. However, sometimes the window wouldn't open. And when the feature worked, there wasn't much feedback to confirm what I was viewing.

CODE AND SYMBOL FORMATS. VTune 3.0 supports C/C++ compilers from Microsoft, Intel, Inprise, Watcom, and IBM; Java compilers from Microsoft, Inprise, and Asymetrix; Delphi, from Inprise; Microsoft Visual Basic; and Intel's Fortran. Most PC games in development are Windows 98 titles using one of these compilers (OK, maybe not the Fortran). Some features aren't available with all compilers. In general, the closer you are to the mainstream of development, the more features are supported.

Alas, VTune won't be of much help in speeding up any legacy tools upon which you may still be relying. I was able to profile a DOS-extended build tool running in a DOS box. Unfortunately, I couldn't get VTune to map the samples to instructions properly, and I had to correlate the results manually. I had even less luck when I tried to statically analyze an object module from a ROM BIOS. VTune can parse COFF object files, but not Intel's own OMF.



WRAPPING IT UP. There's a lot that I didn't discuss here, because Intel had the good sense not to fix what wasn't broken. Instead, they fixed all the bugs that used to frustrate me, added a couple of features from my wish list, and made the product a lot easier to use. This is a mandatory upgrade, and if you don't already own the product, you're in for a treat.

One important feature remains on my wish list: an API. I'd like to be able to write loaders for different module types, monitors for different performance events, and filters for different output formats. I'd like to be able to control VTune's behavior from my program. I'd even like a redistributable run-time module, so I could write a program that monitored its own performance.

VTune 3.0: ★★★★½

Company: Intel Corp.
Santa Clara, Calif.
(800) 253-3696
http://developer.intel.com/design/perftool/vtcd/

Price: $429 ($169 upgrade from any previous version)

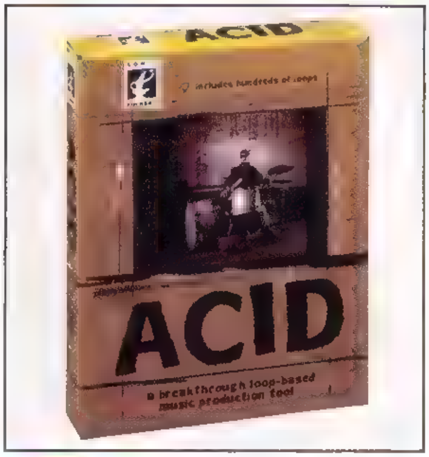
System Requirements: Windows 95 or 98, Windows NT 4.0 (SP3) or NT 5.0 beta 2. Intel Pentium, 32MB RAM, 50MB disk space. The event-based sampling feature requires a Pentium Pro family processor.

Pros:

1. Shows performance information in abundant detail, by instruction, module, process, or processor.
2. New call graph profiling feature counts, times, and charts function calls.
3. Code coaches and online help are excellent teaching tools.

Cons:

1. Needs an API that developers can use to control and extend the product.
2. Data in Chronologies view is hard to correlate with other views.
3. Limited support for non-Windows code running on the Intel architecture.


ACID
includes hundreds of loops
ACID
a breakthrough loop-based music production tool

Sonic Foundry's Acid

by Andrew Boyd

Acid is not, and is not intended to be, a standard multitrack audio environment. Unlike Pro Tools or Cool Edit Pro or even digital audio sequencers, Acid is completely focused on building music from loops. You can't do much in Acid that you couldn't also do in one of these other environments, but if you find yourself making music out of looping audio, you won't find an easier, faster, or more fun program in which to do it. And Acid does offer enough peripheral features to make it flexible. It isn't perfect, but it is very impressive, and it could as easily find a place with a complete novice as with a hardcore professional. Essentially, this is a product you probably didn't know you needed, but that once you've tried, you won't give up.

USING ACID. Acid isn't so much about composing or even editing as it is about assembly. It accepts sound files of three basic types (in several formats) — loops, one-shots, and disk-based — and gives you a set of specialized tools with which to piece them together. Loops and one-shots tend to be short sounds loaded in RAM (Acid provides a RAM monitor so you'll know when you've filled available memory). The main difference between them is that one-shots don't loop — they tend to be crash cymbals, drum hits, vocal snippets, and so on. Disk-based files are often larger files that Acid can get off disk as the project plays. So you might build a rhythm track out of a bunch of drum and bass loops, then load in (or record) a long vocal track to play along with it. That file would be a disk-based stream by default. If for some reason you wanted to load it into RAM, and had enough available, you could override the default settings to do just that.

The program presents a screen split into three adjustable viewing panes: the Track View, the Track List, and a multifunction section that can be set to show an Explorer, an Edit Window, a Mixer, or Effects plug-in pages. The primary pane is the Track View — this is where you put together loops, draw automation curves ("envelopes" in Acid parlance), drop markers, and so on. Though this section looks a lot like a typical multitrack editing environment, its function is pretty different. For instance, a given track can only hold a single sound. It can be any supported type and can have as many instances as you want, but you can't slip a drum fill into a space in a drum loop track to save tracks — the fill will initiate its own track. The down side is that projects build up a lot of tracks very quickly and viewing and navigating through them can become a bit of a chore.

The Track List sits next to the Track View and provides the name of the file in the track, an icon to indicate its type, and some mixing controls (level, pan, effects sends, output assignment, solo, and mute). The bottom pane has a number of modes selectable by standard Windows-style tabs. Most often I used it in Explorer mode, where it provides a standard Windows Explorer view from which to select loops and sounds to load into the project. Another mode allows for "Acidizing" a loop, which involves adding proprietary information, including a root note for transposition, number of beats and/or tempo for time scaling, and so forth. This pane also lets you access a Mixer application, which presents level controls for all wave devices installed in your system (nice — discrete outs for multiple sound cards). Finally, there are tabs to select settings for up to eight DirectX-compatible effects plug-ins (I used some CFX plug-ins installed by Cakewalk Pro Audio 6.01).

The first thing that's noticeable about using Acid is that it has essentially no learning curve. If you've ever done anything with computer audio before and have even a basic idea of how looping works in a musical context, you'll be making music in minutes. The program ships with a number of very usable loops on its CD (additional Loop Libraries from Sonic Foundry are about $60), and these are a great place to start. Set the bottom pane to Explorer view and locate the loops on the CD. As you click on them, they'll automatically play an audible preview. When you find one you like, a double-click will add it to the Track List and create a track for it in the Track View pane (you can also drag and drop it into either pane). Click and drag anywhere in the Track View with the pencil tool (here's one of those really innovative features), and you'll draw a perfectly looped and quantized bit of that sound. Oops, dragged for five bars instead of four! Hold the cursor near the end of this chunk of sound and it becomes a trimming tool. Click and drag back and it will snap to the next bar (or whatever quantization resolution to which you're currently zoomed). On the next track, drop another loop that works well with the first (because tempo and key are matched automatically, nearly everything sounds good together), draw a section of it out, and a song is born. Repeat until done.

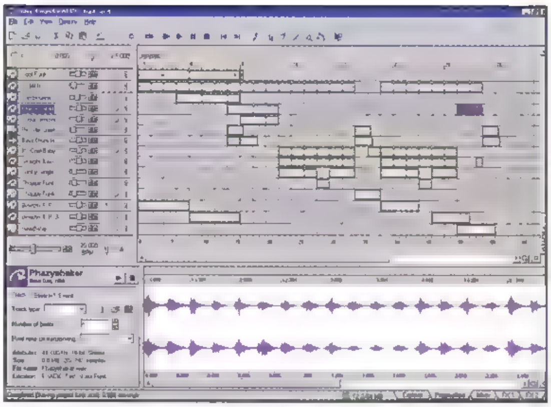
WILL COMPOSE FOR FOOD. Of course, if you really consider yourself a musician or a composer, there's a chance Acid will insult you. Let's face it: it's cheating. Using Acid to compose music is like sculpting with Legos rather than clay. One method requires talent, skill, training, and patience while the

Andrew Boyd has been creating sound for games since 1993. He now runs Audible Images, a music and sound design house in San Francisco, Calif. He can be reached at andrew@audibleimages.com.

other... well, if you have thumbs to grasp the pieces, you're pretty much on your way. Still, schedules and budgets being what they are for game sound, you've probably found yourself building pieces entirely out of samples and loops anyway. Why not make it easier on yourself? You'll be able to put together polished pieces that impress clients in no time, you'll own the license to the music, and you'll even have fun. If you know music and audio, your Acid arrangements will be the better for it, plus you'll work faster and more efficiently.

One potential use for this program in a serious music environment is as a writing or practicing tool. Use it as a super sophisticated drum machine — get a CD full of drum loops, pick out a couple that fit the direction you want to go, and set them to looping. Then you can play your own parts — say on a guitar — over the loops until you get them just right. Want to change the tempo? Just drag the slider (no processing all the loops and reloading them into the project or any of that kind of nonsense). Add a baseline to fill out the groove and you can easily transpose its key too. When your jam sounds good, you can record the guitar part right into Acid along with the rhythm track, and then export it as a sound file to whatever program you use for final production. Acid doesn't really have the features to do a full production, but as an accessory to a more general tool, its simplicity and ease of use can free up time and creative energy.

Which brings me to a few of Acid's limitations and shortcomings. For instance, because each loop gets its own track, screen real estate becomes very precious very quickly. The ability to resize the various viewing panes provides some flexibility, and there is a zoom shortcut menu available through a simple right-click, but the only real solution is to run in absurdly high resolutions. I tested at 1,024×768, and it wasn't nearly enough to get a good view of the piece being edited. I don't have a suggestion for how to show the information better, but the way it is now can get pretty frustrating on a big project.



Also, the mixing functionality is too limited. For instance, while eight effects sends is pretty generous, because individual tracks don't have even simple EQ controls, if you want EQ (and to mix a lot of premade loops well, you will) you'll have to start burning through those sends. Because the source material is usually a bunch of little files, it's easy and fast enough to pop them into your wave editor (a single button-click), EQ them, and reopen them in Acid. But given the clean, simple approach of the rest of the program, it doesn't seem as though that should be necessary for so fundamental a process.

PERFORMANCE. Sonic Foundry has obviously got some real throbbing-brain types engineering its stuff. Acid's performance is stunning. My test machine ran Windows 95 on a not-very-impressive-anymore Pentium 200MMX with 64MB RAM and a Turtle Beach Pinnacle sound card. Acid was always quite responsive, even while playing huge projects with time and pitch scaling and a couple of effects plug-ins running. Specifically, I noted the following: on a 15-track piece with no time or pitch modulation and a CFX two-band EQ on FX1 and a CFX Reverb on FX2, the program used about 50 percent of available processor power, occasionally peaking as high as 70 percent. Enabling some time and pitch scaling added around 5 to 10 percent to these numbers. But on a current, fast machine you should run out of the desire to add loops or effects before you run out of horsepower.

WHAT A TRIP. When I first heard about Acid, I thought it seemed like a good idea, but I really didn't see the point. I've built plenty of pieces of music out of licensed loops and samples, and I have plenty of tools with which do it. But as I started playing with this tool, I thought about how many hours of my life have been sunk into searching around for that perfect loop at the right tempo, trying to pitch-adjust all the loops in a song individually so they'll work in the right key, and so on. Acid does what it does so well, and is just so much fun to use, that I have little doubt it will soon be a standard audio tool. ■

Sonic Foundry's Acid: ★★★★

Company: Sonic Foundry
Madison, Wis.
(800) 57 SONIC
http://www.sonicfoundry.com

Price: $399

System Requirements: An Intel Pentium 133 or Alpha AXP microprocessor, Microsoft Windows 95 or Windows NT 4.0 or later, a Windows-compatible sound card, a VGA display, a CD-ROM drive, 32MB RAM, and 5MB hard-disk space for program installation.

Pros:

1. Very fast and transparent way to assemble loop-based productions.
2. An interface so intuitive it's actually fun to use.
3. Excellent sounding (and efficient) real-time pitch and time scaling algorithms.
4. Ability to utilize discrete outputs is a nice touch.

Cons:

1. Somewhat inadequate mixing functionality.
2. Limited built-in wave editing.
3. Homemade loops require more fine-tuning than claims indicate.
4. As fun as it is, it's a bit pricey for a toy — make sure you actually have a use for it.

NETIMMERSE

Develop your game play, not your game engine.

The Fastest Path from 3D Game Conception to Completion

Screen shot from Darkstorm™ by Mythic Entertainment LLC

NETIMMERSE is a professionally engineered, C++ 3D game engine that allows you to **begin developing content from Day 1. Save 6 months of development time** and **years of engineering costs** right out of the gate. Free your software programmers for the **really cool stuff** that game players see on the screen.

Here are some of the reasons why game developers such as **Red Orb Entertainment, GreyStone Technology, MediaStation, Mythic Entertainment, [illegible] Entertainment** and others rely on NETIMMERSE:

- **It delivers action -** aggressive culling, portal-based visibility, and hand optimization in critical code
- **It delivers realism -** multi-textures, projected textures, animated textures, environment maps, and a wide variety of lighting effects
- **It delivers advanced animation -** morphing vertices, particle systems, and quaternion-based keyframes
- **It delivers scene management -** dynamic collision detection, continuous level of detail, and 3D picking
- **It's compatible -** provides a plug-in to 3D Studio MAX®, MultiGen's Creator® and Motion Factory's Motivate™, and works with Direct3D, OpenGL and Glide
- **It makes business sense -** licenses include access to full source code, no royalties, and complete support

Contact NDL today to get on the 3D game fast track!

Check out The Business Case for Buying Instead of Building a 3D Game Engine on NDL's Web site.

www.ndl.com

sales@ndl.com

Palo Alto, CA Sales Office - tel: 650-328-4388; fax: 650-328-3454

Chapel Hill, NC Headquarters - tel: 919-929-2917; fax: 919-967-3237

Image from Prince of Persia® 3D, courtesy of Red Orb Entertainment

Making Kine More Flexible

Two months ago ("Oh My God, I Inverted Kine!" September 1998), I left off discussing methods for real-time inverse kinematics. That column was meant to be an introduction to the idea of inverse kinematics, but mistakes I made proved to be quite a lesson in trigonometry and optimization.

I thought I would start off by taking another look at that column and what I have learned since. There were a couple of typographical errors that I missed. First off, there was a mistake in the forward kinematic definition for the position of the origin of the second bone, Equation 1.

It was written as

$$\theta_2 = (L_1 * \cos(\theta_1), L_1 * \sin(\theta_1)) \qquad \text{Eq. 1.}$$

The use of the theta symbol implied that the result would be an angle. That was an error. The equation should have showed the origin of bone 2 or

$$O_2 = (L_1 * \cos(\theta_1), L_1 * \sin(\theta_1)) \qquad \text{Eq. 1 (revised).}$$

The second typo was in Equation 6. This is the inverse kinematic solution for the angle θ_1. The equation for that angle should have matched what's in the accompanying source code, and read

$$\theta_1 = \frac{(L_1 * \sin(\theta_2))x + (L_1 + L_2 * \cos(\theta_2))y}{(L_1 * \sin(\theta_2))y + (L_1 + L_2 * \cos(\theta_2))x} \qquad \text{Eq. 6.}$$

The above formula matches what is actually being done in the sample code. But wait, there's still trouble with that equation — it turns out that it's wrong. Eran Gottlieb wrote to me to point out that he believed there was an error in the Watt and Watt *Advanced Animation and Rendering Techniques* book that I was using for a source. He believed that there was an atan() missing from the equation, and offered a fairly complicated algebraic proof to back it up. I looked at the geometric solution I also had derived in September (but didn't code up) and it did seem as though something was wrong. So, trying yet another method, I took a look at what was happening. Start off by looking at Figure 1.

From this image, I can determine several things:

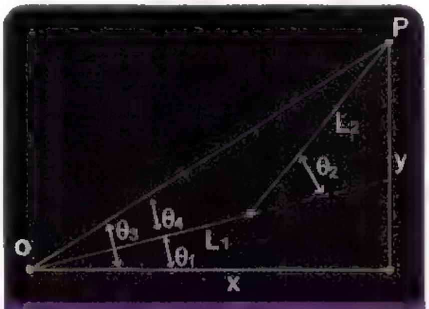


FIGURE 1.

$$\tan(\theta_3) = y/x$$

$$\tan(\theta_4) = \frac{L_2 * \sin(\theta_2)}{L_2 * \cos(\theta_2) + L_1}$$

$$\theta_1 = \theta_3 - \theta_4$$

Now use the tan identity,

$$\tan(a - b) = \frac{\tan(a) - \tan(b)}{1 + \tan(a)\tan(b)}$$

and do some substitution:

$$\tan(\theta_1) = \frac{\frac{y}{x} - \frac{L_2 * \sin(\theta_2)}{L_2 * \cos(\theta_2) + L_1}}{1 + (\frac{y}{x} * \frac{L_2 * \sin(\theta_2)}{L_2 * \cos(\theta_2) + L_1})}$$

If I multiply out the x and the value for $\tan(\theta_4)$, I get

$$\tan(\theta_1) = \frac{y(L_2 * \cos(\theta_2) + L_1) - x(L_2 * \sin(\theta_2))}{x(L_2 * \cos(\theta_2) + L_1) + y(L_2 * \sin(\theta_2))}$$

$$\theta_1 = \arctan(\frac{y(L_2 * \cos(\theta_2) + L_1) - x(L_2 * \sin(\theta_2))}{x(L_2 * \cos(\theta_2) + L_1) + y(L_2 * \sin(\theta_2))})$$

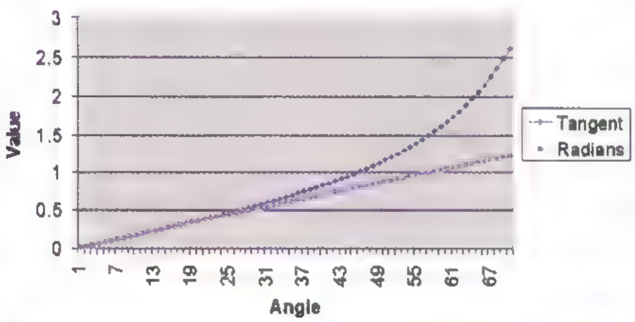


FIGURE 2. *Tangents vs. Radians.*

When not stuck with his head in some geeky book, Jeff can be found at the local brew-pub. Try to catch him before the lunch special at jeffl@darwin3d.com.

DEVELOPMENT ENVIRONMENTS • DEVELOPMENT LIBRARIES/APIS/ENGINES • DEVELOPMENT UTILITIES • GRAPHICS CARDS • MODELING & ANIMATION ENVIRONMENTS • SOUND CARDS
MODELING & ANIMATION PLUG-INS • IMAGE EDITING & MANIPULATION • VIDEO EDITING & COMPOSITING • RECORDING & EDITING • COMPOSITION • PRODUCTION UTILITIES

Readers:

ROCK THE VOTE

Choose your favorite game development products by filling out and mailing the attached ballot.*

IT'S EASY.
IT'S FUN.
WE'RE PICKING UP THE POSTAGE.

Winners will be announced at the Game Developers Conference March 15-19, 1999 at the San Jose Convention Center. Can you stand the suspense?

PREFERRED PRODUCT AWARD
Game DEVELOPER
1999

Game DEVELOPER

Ballots MUST be postmarked by December 1, 1998. Eligible products must be released since July 1, 1997.

Thanks for not being apathetic.

* To prevent unscrupulous individuals and renegade user groups from stuffing the ballot box by email or fax, only original ballots sent via snail mail will be counted.

Game DEVELOPER

So, now I have further proof that there should have been an atan() in my Equation 6 from September. However, part of the reason I create demo applications to go with each article is that I want to make sure that everything is working. The demo application from September seemed to work fine — or did it? When I played with it a bit more, I noticed that the solution point seemed to diverge from the desired position as I approached the bottom of the screen. This clued me in on what was happening. Measured in degrees, the tangent of an angle hardly resembles the actual angle at all. But since, in the program, all of the calculations are handled in radians, something interesting happens. Take a look at Figure 2.

This is pretty interesting. For angles less then 30 degrees or so, the tangent of the angle is approximated by the radian values for those angles. This is the reason everything seemed to work in my demo program. When the total range of freedom is in the range of 0 to 90 degrees and the value of the tangent of the angle is less then 0.5, the atan() step can be eliminated. That could be very helpful for a limited application such as mine. Because of the symmetrical nature of the tangent, some additional checks would be needed to handle a joint with greater freedom. But for game programmers looking to squeeze out speed anywhere they can, this could be a nice hack. Anyway, I fixed the code and updated the source on the *Game Developer* web site (www.gdmag.com), so you can play around with it yourself. Now that I finally have the analytical solution all neatly worked out, we can tuck that away until later

Iterative Methods for Complex Inverse Kinematics

As I mentioned in the September column, analytical solutions for inverse kinematic problems are desirable because of their speed and exactness of solution. However, for complex kinematic problems, an analytical solution may not be possible. Simply adding another joint to the system greatly increases the complexity. Because my goal is to be able to solve any inverse kinematic problem I may create, a more general solution is necessary. To solve these more general problems, I need to turn to iterative methods. In an iterative solution, small adjustments are made to the joints to solve the inverse kinematics in a series of steps. This process is finished when the end effector reaches the goal within some tolerance.

Analytical solutions for inverse kinematic problems are desirable because of their speed and exactness of solution. However, for complex kinematic problems, an analytical solution may not be possible.

The robotics community has established methods for solving the inverse kinematics of an arbitrary system. The solutions are generally based on either matrix inversion techniques or some form of optimization. Matrix inversion is a complicated process that is both computationally very expensive, and plagued by a variety of other problems that stem from numerical instabilities. Optimization-based methods avoid matrix inversion completely. They also attempt to minimize the error in the system. In the case of inverse kinematics, one method would be to minimize the distance between the goal point and the end effector of the chain. This can be accomplished by adjusting the joint angles in a way that minimizes this distance.

While there's plenty of research on approaches using both techniques, it seemed to me that the optimization-based methods might be more likely to produce the real-time results I desired.

Cyclic-Coordinate Descent

In his masters thesis on inverse kinematics (see "For Further Info"), Chris Welman described a method for solving these problems using a technique termed cyclic-coordinate descent (CCD). This technique was first outlined by Li-Chun Tommy Wang and Chih Cheng Chen in a paper in *IEEE Transactions on Robotics and Automation*. CCD involves minimizing the system error by adjusting each joint angle one at a time. The method he used starts at the last link in the chain and works backwards, adjusting each joint along the way. Take a look at Figure 3A.

I start with the last link in the chain. First, I create a vector from the root of the current link, R, to the current effector position, E. Another vector is created from R, to the desired endpoint, D. I wish to determine the value of angle, A, in order to rotate the vector RE onto vector RD. This is where a technique that is used quite a bit in 3D games comes in handy. There are often times in a game where you want to calculate the angle by which a player needs to rotate to face an opponent. I can use the same method to solve my IK problem. The

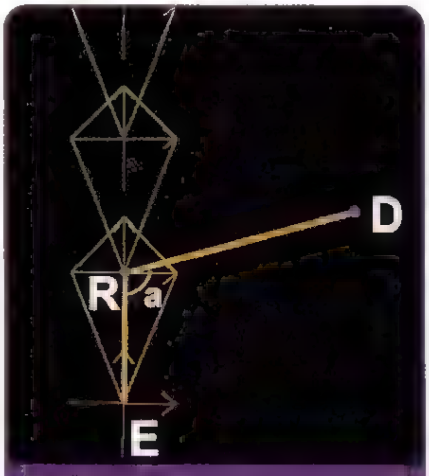


FIGURE 3A.

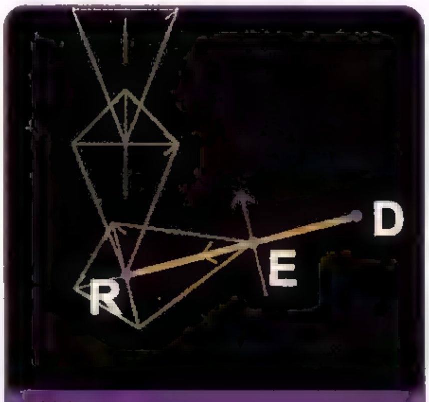


FIGURE 3B.

dot product relationship between two vectors is defined as

$$\cos(a) = \underset{RD}{\longrightarrow} \bullet \underset{RE}{\longrightarrow}$$

By taking the inverse cosine of the dot product, I get the angle between the vectors.

However, because the dot product only tells me the angle, I also need to know the direction I need to rotate about R. For this, I can turn to the cross product. The cross product operation creates a vector that is perpendicular to the two vectors. By checking the sign of the z element of the vector, I know which way to rotate. This is the amount by which I modify the joint. You can see the results in Figure 3B. I then move one link up the chain and repeat the process, as you can see in Figures 3C and 3D.

This continues up the chain until the base joint is reached, and then the process is repeated, starting at the last joint again. This process is repeated until either the end effector is close enough to the desired position or the loop has repeated a set number of times. (This break count is needed to allow for positions that are not reachable.)

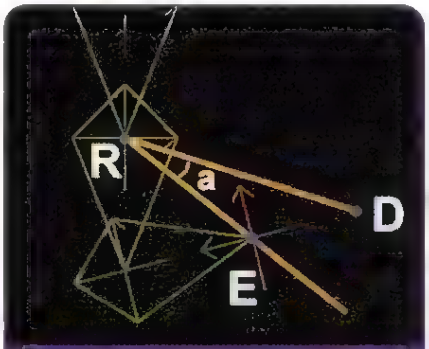


FIGURE 3C.

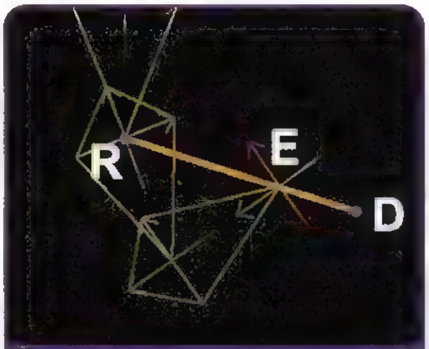


FIGURE 3D.

LISTING 1. The CCD Algorithm (continued on page [illegible]).

```
#define EFFECTOR_POS   5    // THIS CHAIN HAS 5 LINKS
#define MAX_IK_TRIES   100  // TIMES THROUGH THE CCD LOOP
#define IK_POS_THRESH 1.0f // THRESHOLD FOR SUCCESS
///////////////////////////////////////////////////////////////////////////////
// Procedure: ComputeCCDLink
// Purpose:   Compute an IK Solution to an end effector position
// Arguments: End Target (x,y)
// Returns:   TRUE if a solution exists, FALSE if the position isn't in reach
///////////////////////////////////////////////////////////////////////////////
BOOL COGLView::ComputeCCDLink(CPoint endPos)
{
/// Local Variables ///////////////////////////////////////////////////////////
tVector  rootPos,curEnd,desiredEnd,targetVector,curVector,crossResult;
double   cosAngle,turnAngle,turnDeg;
int      link,tries;
///////////////////////////////////////////////////////////////////////////////
// START AT THE LAST LINK IN THE CHAIN
link = EFFECTOR_POS - 1;
tries = 0; // LOOP COUNTER SO I KNOW WHEN TO QUIT
do
{
// THE COORDS OF THE X,Y,Z POSITION OF THE ROOT OF THIS BONE IS IN THE MATRIX
// TRANSLATION PART WHICH IS IN THE 12,13,14 POSITION OF THE MATRIX
    rootPos.x = m_Link[link].matrix.m[12];
    rootPos.y = m_Link[link].matrix.m[13];
    rootPos.z = m_Link[link].matrix.m[14];

    // POSITION OF THE END EFFECTOR
    curEnd.x = m_Link[EFFECTOR_POS].matrix.m[12];
    curEnd.y = m_Link[EFFECTOR_POS].matrix.m[13];
    curEnd.z = m_Link[EFFECTOR_POS].matrix.m[14];

    // DESIRED END EFFECTOR POSITION
    desiredEnd.x = endPos.x;
    desiredEnd.y = endPos.y;
    desiredEnd.z = 0.0f;      // ONLY DOING 2D NOW

    // SEE IF I AM ALREADY CLOSE ENOUGH
    if (VectorSquaredDistance(&curEnd, &desiredEnd) > IK_POS_THRESH)
        {
    // CREATE THE VECTOR TO THE CURRENT EFFECTOR POS
    curVector.x = curEnd.x - rootPos.x;
    curVector.y = curEnd.y - rootPos.y;
    curVector.z = curEnd.z - rootPos.z;

    // CREATE THE DESIRED EFFECTOR POSITION VECTOR
    targetVector.x = endPos.x - rootPos.x;
    targetVector.y = endPos.y - rootPos.y;
    targetVector.z = 0.0f;      // ONLY DOING 2D NOW

    // NORMALIZE THE VECTORS (EXPENSIVE, REQUIRES A SQRT)
    NormalizeVector(&curVector);
    NormalizeVector(&targetVector);

    // THE DOT PRODUCT GIVES ME THE COSINE OF THE DESIRED ANGLE
    cosAngle = DotProduct(&targetVector,&curVector);

    // IF THE DOT PRODUCT RETURNS 1.0, I DON'T NEED TO ROTATE AS IT IS 0 DEGREES
    if (cosAngle < 0.99999)
    {
    // USE THE CROSS PRODUCT TO CHECK WHICH WAY TO ROTATE
```

WHAT COULD 30 MILLION PEOPLE POSSIBLY SEE IN ATI...
OPTIMIZE FOR ATI AND YOU HAVE [illegible] YOU INSTANTLY
GRAPHICS BY ATI rage 128
ATI®
www.atitech.com/gd

© Copyright 1998 ATI Technologies Inc. ATI and RAGE 128 are trademarks and/or registered trademarks of ATI Technologies Inc. All other company and/or product names are trademarks and/or registered trademarks of their respective manufacturers. Products may not be exactly as shown. Features, performance and specifications may vary depending on operating environment and are subject to change without notice.

LISTING 1, CONT.

```
  CrossProduct(&targetVector, &curVector, &crossResult);
  if (crossResult.z > 0.0f) // IF THE Z ELEMENT IS POSITIVE, ROTATE CLOCKWISE
    {
   turnAngle = acos((float)cosAngle);  // GET THE ANGLE
   turnDeg = RADTODEG(turnAngle);              // COVERT TO DEGREES
   // DAMPING
   if (m_Damping && turnDeg > m_Link[link].damp_width)
     turnDeg = m_Link[link].damp_width;
   m_Link[link].rot.z -= (float)turnDeg;       // ACTUALLY TURN THE LINK
   // DOF RESTRICTIONS
   if (m_DOF_Restrict &&
     m_Link[link].rot.z < (float)m_Link[link].min_rz)
     m_Link[link].rot.z = (float)m_Link[link].min_rz;
     }
   else if (crossResult.z < 0.0f)      // ROTATE COUNTER CLOCKWISE
     {
   turnAngle = acos((float)cosAngle);
   turnDeg = RADTODEG(turnAngle);
   // DAMPING
   if (m_Damping && turnDeg > m_Link[link].damp_width)
    turnDeg = m_Link[link].damp_width;
   m_Link[link].rot.z += (float)turnDeg;       // ACTUALLY TURN THE LINK
   // DOF RESTRICTIONS
   if (m_DOF_Restrict &&
    m_Link[link].rot.z > (float)m_Link[link].max_rz)
    m_Link[link].rot.z = (float)m_Link[link].max_rz;
    }
   // RECALC ALL THE MATRICES WITHOUT DRAWING ANYTHING
   drawScene(FALSE);  // CHANGE THIS TO TRUE IF YOU WANT TO SEE THE ITERATION
   }
  if (--link < 0) link = EFFECTOR_POS - 1;       // START OF THE CHAIN, RESTART
  }
 // QUIT IF I AM CLOSE ENOUGH OR BEEN RUNNING LONG ENOUGH
 } while (tries++ < MAX_IK_TRIES &&
  VectorSquaredDistance(&curEnd, &desiredEnd) > IK_POS_THRESH);
 return TRUE;
}
```

Implementation

The algorithm I've just outlined is a simple form of the CCD method. I only was concerned with the position of the final effector. A true IK system would also allow you to give a target orientation for the end effector. This can be added by changing the test for the error in the system and the amount of adjustment needed. For my simple needs, the position goal was enough. To measure the amount of error in the system, I checked the squared distance between the current end effector and the desired position. By using the squared distance, I avoid the added computational cost of an extra square root.

It is very easy to get this algorithm up and running in OpenGL. The matrix stack makes it easy to keep track of individual transformations at each joint in the hierarchy. During the recursive transformation routine, the matrix is grabbed and stored. The translation portion of that rotation matrix holds the position of the root of that joint. So creating the vectors is pretty easy. I also needed some special code to handle when the dot product produced a value of 1. This happened when the angle between the two vectors was zero degrees, a point at which I don't want anything else to happen.

I turned off the drawing while the algorithm was running, but if you leave it on you can see the steps as they progress. In an animation application, only the final solution angles are used as keyframes for a quaternion interpolation. That way, the animation would be quite smooth. The complete code for the CCD algorithm is in Listing 1.

Improving the Method

RESTRICTIONS ON DEGREES OF FREEDOM. In many character hierarchies you may want to impose limits on the degrees of freedom (DOF) of an individual joint. This would keep an individual joint from rotating into a position that is physically impossible for a character to achieve. In some other inverse kinematic methods, this can be a bit complicated. However, in the CCD method, such restrictions are easy. Because of the way the method handles each joint as a single analytical geometry problem, any limits on individual joints are simply figured into the problem. When the routine goes to update the joint rotation, a test determines if the joint is outside the limits. If it is, the joint is clamped to those limit angles. The rest of the joints are then used to satisfy the problem during later steps.

DAMPING. The CCD method will rotate an individual joint to any angle needed to satisfy the problem at any step. Because the routine starts from the last joint and works in, the method tends to favor later joints. This bias may not always look natural. Further, because each joint can swing wildly at each step, kinks are sometimes present in the resulting chain. By limiting the amount a single joint angle can change at each step, both of these effects can be controlled.

You can see the effect that DOF restrictions and damping have on the algorithm in Figure 4A and 4B. Figure 4A shows the algorithm without any restrictions and Figure 4B has both restrictions turned on. However, you really have to play with it interactively to get a real feel for how these improvements change things.

Conclusions

I now have a robust system for solving inverse kinematic problems of any number of links. The iterative nature of the algorithm makes it both easy to control and simple to modify. In fact, with the CCD algorithm, adding extra links is no more difficult to set up. It is simply an extra step in the iteration. Restrictions on joints can be easily added to enhance the realism.

The solver is still 2D; the algorithm also work in 3D, but the error correction would need to be changed around

1978
1988
1998

your titles deserve

QUALITY DIGITAL VIDEO

TrueMotion® 2.0
Compression Tool Kit

[illegible]

COME SEE FOR YOURSELF

www.duck.com
E-mail us: games@duck.com

to rotate about the perpendicular angle.

By only optimizing for the final position of the end effector, things are much simpler. If you are concerned with the orientation of the final effector also, the solution is more difficult. However, for many 3D real-time applications, the former methods work great.

Sample Application

The sample application this month allows you to interact with the IK solver. You click and drag on the screen and the snake attempts to reach your mouse. You can toggle the damping and DOF restrictions on and off as well as adjust their settings. More ambitious readers may want to play around with it by adding links, adding the orientation optimization, and perhaps converting the routine to 3D. I think there are many possibilities using this technology in real-time gaming, and programmers are just beginning to explore these areas.

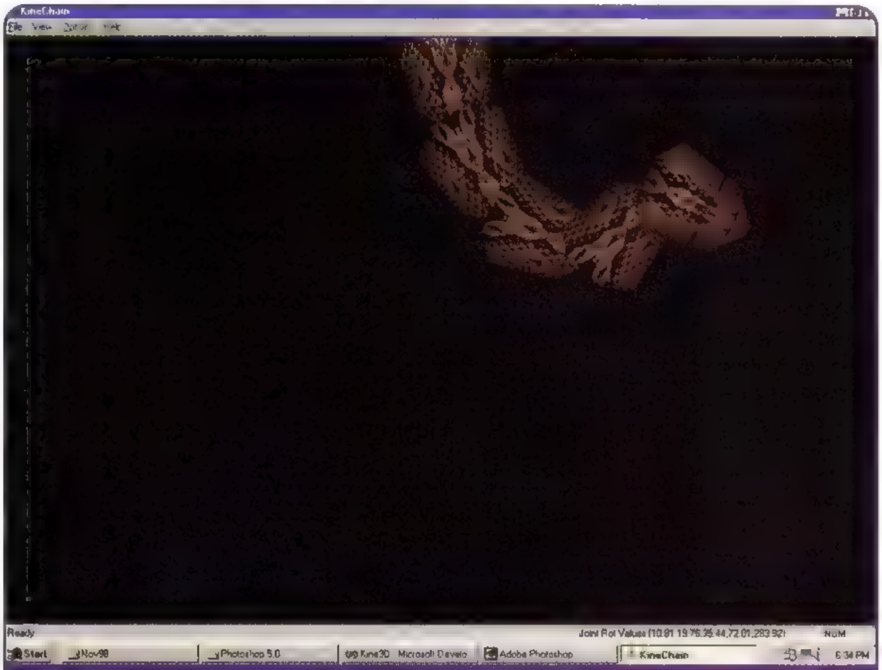

KineChain
File
View
Option
Help
Ready
Joint Rot Values (10.81 19.76.35.44,72.01,283.92)
NUM
Start
Nov98
Photoshop 5.0
Kine3D
Microsoft Develo
Adobe Photoshop
KineChain
6:34 PM

FIGURE 4A. The effects of the algorithm without any limiting terms (DOF or Damping).

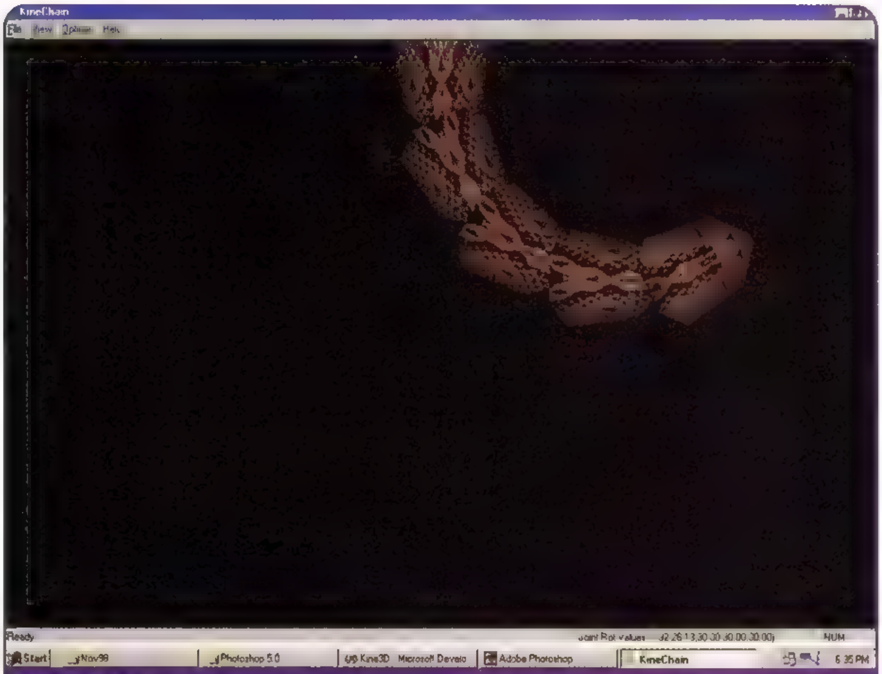

KineChain
File
View
Option
Help
Ready
Joint Rot Values (32.26 13,30.00 30.00,30.00)
NUM
Start
Nov98
Photoshop 5.0
Kine3D
Microsoft Develo
Adobe Photoshop
KineChain
6:35 PM

FIGURE 4B. The effects of the algorithm when limiting the DOF and employing damping.

Acknowledgments

Thanks go out to Eran Gottlieb for catching the error in the September code and forcing me to take a deeper look at it. Also, thanks to Dr. Alan Watt for a prompt confirmation of the bug in the book as well as a deeper insight into what he intended. Dr. Watt is currently working on a book on 3D game programming. I am sure everyone out there will eagerly await that book. ■

[illegible]

- Welman, Chris. *Inverse Kinematics and Geometric Contraints for Articulated Figure Manipulation.* Masters Thesis, Simon Fraser University, 1993. This paper was a gold mine of references for me as well as a very good overview of the issue. Chris worked on the well-known Lifeforms animation system with Thomas Calvert. He now works on graphics software at Mainframe Entertainment, the makers of *Reboot* and *Beast Wars*. This paper is available on the web at: http://fas.sfu.ca/pub/cs/theses/1993/ChrisWelmanMSc.ps.gz
- Wang and Chen. "A Combined Optimization Method for Solving the Inverse Kinematics Problem of Mechanical Manipulators." *IEEE Transactions on Robotics and Automation*. Vol. 7, No. 4, August 1991, pp. 489-499. Original outline of the CCD method.
- Wright and Sweet. *OpenGL Super Bible*. Corte Madera, Calif.: Waite Group Press, 1996. I used this for the basis for some of the OpenGL code in the sample application.

cakewalk
Pro Audio
24-bit/96 kHz
8
00:06
09:00
Vector Automation
Real-time
MIDI Plug-ins
Sync
AVI
MPEG
QuickTime
Now Shipping
The new version of the leading MIDI and digital audio software
www.cakewalk.com
888-CAKEWALK
cakewalk
All trademarks mentioned are held by their respective owners.

San Jose Convention Center | **Expo:** March 16-18 | **Conference:** March 15-19

Nihilistic is bringing the entire team to the GDC '99.

(They may have to take three cars.)

nihilistic software

L to R Back Row:
Yujin Kiem Artist
Maarten Kraaijvanger Lead Artist
Rick Servande Artist
Steve Tietze Director of World Design | Lead Level Designer
Rob Huebner VP | Director of Technology | Lead Programmer

L to R Front Row:
Brent McLeod | Level Designer
Steve Thoms | Level Designer
Ingar Shu | Sound Designer | Programmer
Ray Gresko | President | Producer | Designer | Programmer | Business Manager
Yves Borckmans | Programmer
Anthony Chiang | Technical Artist

Ray Gresko, lead programmer and co-designer of LucasArts' JEDI KNIGHT: DARK FORCES II™, comes back year after year to the Game Developers Conference. In 1999, every member of his new company, Nihilistic Software, is coming too. They'll lead sessions, learn from over 250 industry experts, network with thousands of their peers, and have a great time doing it! Maybe they'll find a new tool on the expo floor to speed production of their 3D RPG shipping next fall. What they'll take back to the office from the Game Developers Conference will help them make better games and ship them on time. It's the three days this young, fast-growing company can't afford to miss

The Game
Development Platform
for Real Life

GAMEDevelopers CONFERENCE 1999

San Jose Convention Center **Expo:** March 16-18 **Conference:** March 15-19

(**To Register**)

w: www.gdconf.com/mgdm **t:** 888.234.9476
f: 415.905.2222 **e:** register@mfgame.com

Miller Freeman Game Group · Game Developer · Gamasutra.com · GAMEDevelopers Conference

by Mel Guymon

Painting With Vertex Colors

Yet another weapon in the arsenal of today's 3D artists, vertex colors allow us to get more bang for our pixel buck without spending an iota of texture space. Most of today's cutting edge 3D engines support lighting of some sort, either dynamic or precalculated, and in many cases, this is done through the use of color vertices. Color vertices allow us to give life to our environments through the use of light and shadow, and also allow us to alter the colors of our 3D models without changing the textures or mapping coordinates. Although the technology for using color vertices has been around almost as long as we've been rendering polygons, the application and widespread use of the techniques has been largely ineffective due to limited on-screen polygon counts. This is yet another example of how the increase in rendering power of today's hardware accelerators has an unexpected side-benefit.

This month we'll look at three examples of how to use color vertices to our advantage, and also talk about the tools and technology that make these changes possible.

Terms and Definitions

COLOR VERTEX. In many real-time 3D engines, the RGB value of each vertex is stored right along with the geometry. In most cases, this value is normalized to somewhere in the lighter end of the spectrum (on a scale from 0 to 255, values of 175, 175, 175, or about 2/3 the distance between black and white). If at some point in the data generation process these values can be altered, we can add and adjust the colors in our geometry without spending any of our valuable texture space.

GENERAL COLOR VERTEX APPLICATION. Color vertices derive their color through a combination of up to three basic processes:

- Color vertices can be modified by hand, by directly choosing the color of each vertex
- Color vertices can be procedurally generated through lighting
- Color vertices can be procedurally generated through texture sampling.

While there are many capable programs on the market that support and allow modification of color vertices, we'll only be looking at two, 3D Studio MAX 2.5 and Softimage 3.8.

VERTEX COLOR LIMITATIONS. The main limitation when working with color vertices is that you need to have a vertex in order to have vertex color. This common-sense notion comes into play particularly when you use color vertices to add areas of shadow to your environments. Typically, large flat floors are made up of a minimum number of polygons simply for efficiency. Yet, as Figure 1 shows, trying to use color vertices to represent cast shadows requires that you have sufficient vertices to add color. You may find yourself doing some tesselating in certain areas simply to get enough vertex resolution.

BASIC COLOR THEORY. There are many excellent references on this topic, and it helps to have an idea of what works before sitting down to add lighting to a scene. Here are some very general

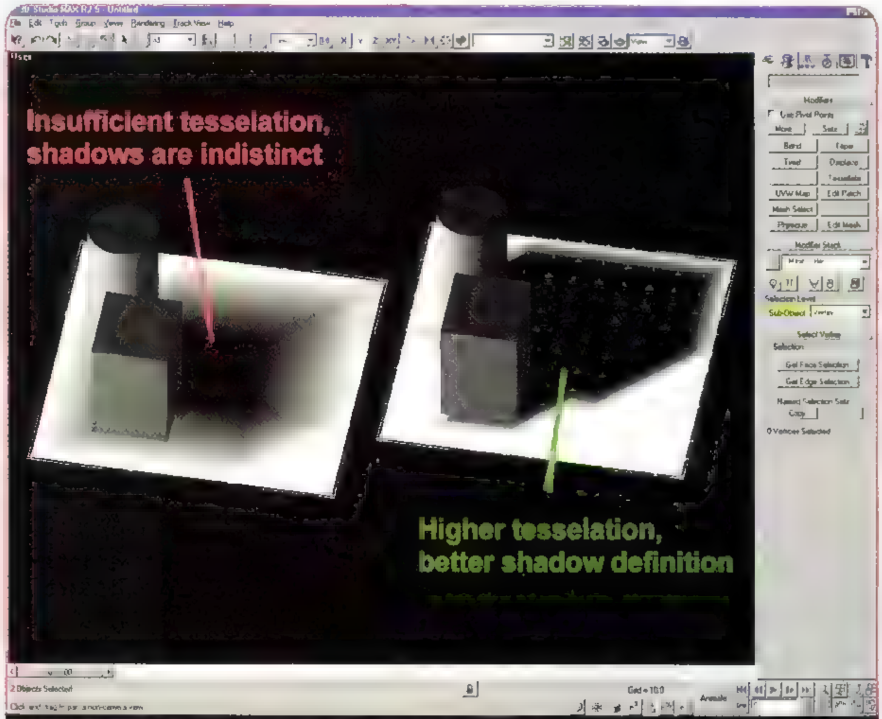

Insufficient tesselation, shadows are indistinct
Higher tesselation, better shadow definition

FIGURE 1. *When using color vertices to represent cast shadows, you must have a sufficient number of vertices to add color.*

Mel has worked in the games industry for several years, with past experience at EIDOS and Zombie. Currently, he is working as the art lead on DRAKAN (www.surreal.com). Mel can be reached via e-mail at mel@surreal.com.

principles to remember when working with color:

- Colors affect emotion. Reds are hot and elicit excitement and restlessness, while cool blues and warm greens tend to engender feelings of happiness and calm.
- Color affects perception. Objects that are lighter in color appear larger, especially when placed in dark spaces. The flip side is true as darker objects appear smaller.
- Complementary colors contrast. The boundary between complementary colors acts as a focal point; choose yellow lights to highlight areas in a blue-lit environment, red for cyan, and so on.
- Color balance is key. Subtle lighting effects are usually more effective in setting the mood; try to stick with two or three colors in each scene.

Environmental Lighting

In this case, we'll look at environmental lighting with procedural color generation through raytraced lighting. Most 3D engines support real-time dynamic lighting using simple point lights in 3D space. These lights can greatly enhance the mood of an environment, but can be computationally expensive. In this example, we look at how to create a lighted environment in Softimage without using any real-time lights.

Although animation is probably Softimage's real strength, Softimage does have a rather robust native modeling and texturing package, as well as a color vertex utility called RenderVertexColors. This aptly named plug-in, found in the Matter module, allows the user to "bake" the colors into the vertices based on the material properties and lighting in the scene. It's actually as simple as it sounds; you select the object whose vertices you want to color, then you execute the RenderVertexColors utility. The program then renders the object internally, and determines the color of each vertex based on the lighting and/or

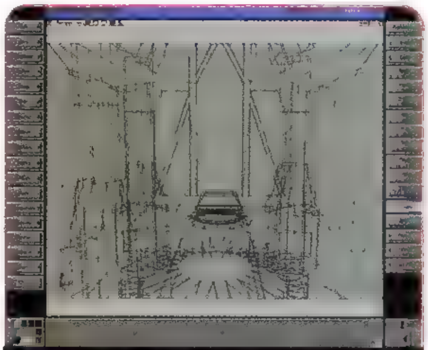


FIGURE 2. *Wireframe-rendered environment in Softimage 3.8.*

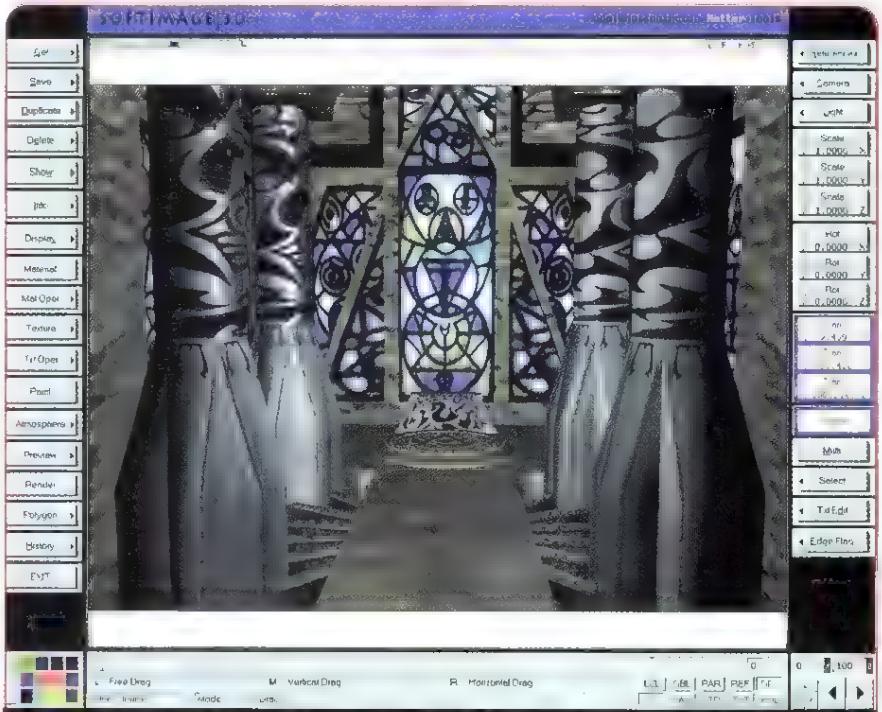

SOFTIMAGE|3D
Get
Save
Duplicate
Delete
Show
Info
Display
Material
Mat Oper
Texture
Tx Oper
Paint
Atmosphere
Preview
Render
Polygon
History
EXIT
Camera
Light
Scale
Rot
Multi
Select
Txt Edit
Edge Flag
Free Drag
Vertical Drag
Horizontal Drag

FIGURE 3. *The wireframe scene after the application of textures and a generic lighting model.*

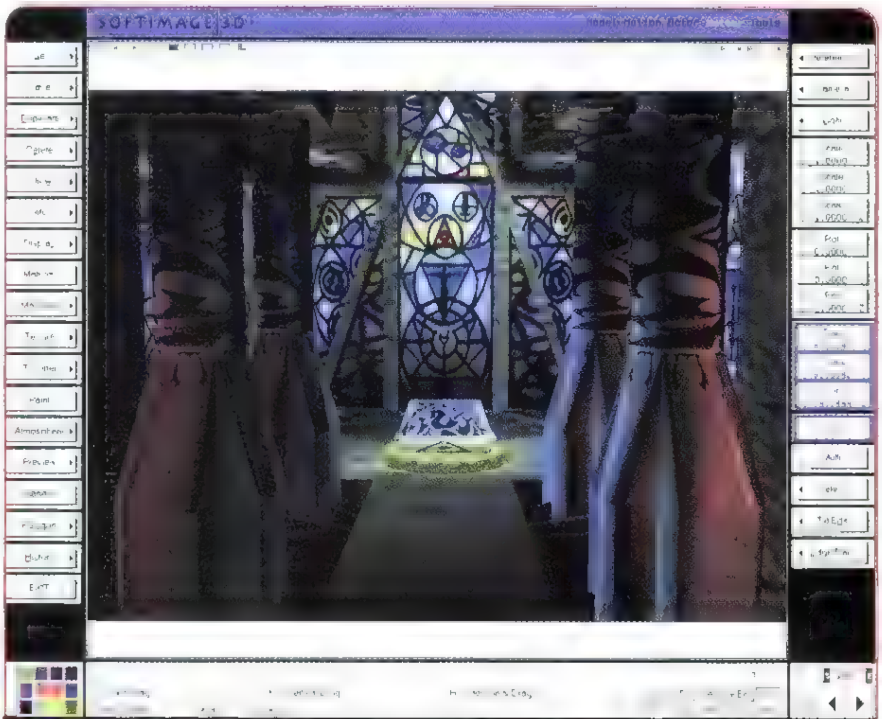

SOFTIMAGE|3D
Atmosphere
Preview
EXIT

FIGURE 4. *The scene was lit using several point lights, and then the lights were removed, so that the lighting values remain, stored in the vertices.*

texturing of the object. The vertex is then assigned that color, and the data is hardcoded into the geometry. The lights can now be removed from the scene and the geometry exported to the 3D engine.

In Figure 2, we have a Softimage wireframe of a scene in a real-time 3D game. Figure 3 shows this same scene with a generic lighting model and textures applied (this is similar to what we usually see in most 3D engines: a statically lit world with some areas of color). Figure 4 shows the result of lighting the scene with several point lights, removing those lights, and then storing the lighting values "in the vertices." Note that to light this single scene in real-time with eight to ten lights would bring most 3D engines to a shuddering halt, yet we've accomplished the same result without the use of a single real-time light.

SUGGESTIONS FOR IMPLEMENTATION. Leave the lighting alone until the final pass. Taking the time to get the lighting right can make the difference between a really stunning environment and one that's just run of the mill. But this time is often wasted if the geometry needs to be modified for game play or polygon-count considerations. Don't be afraid to use primary colors. A good

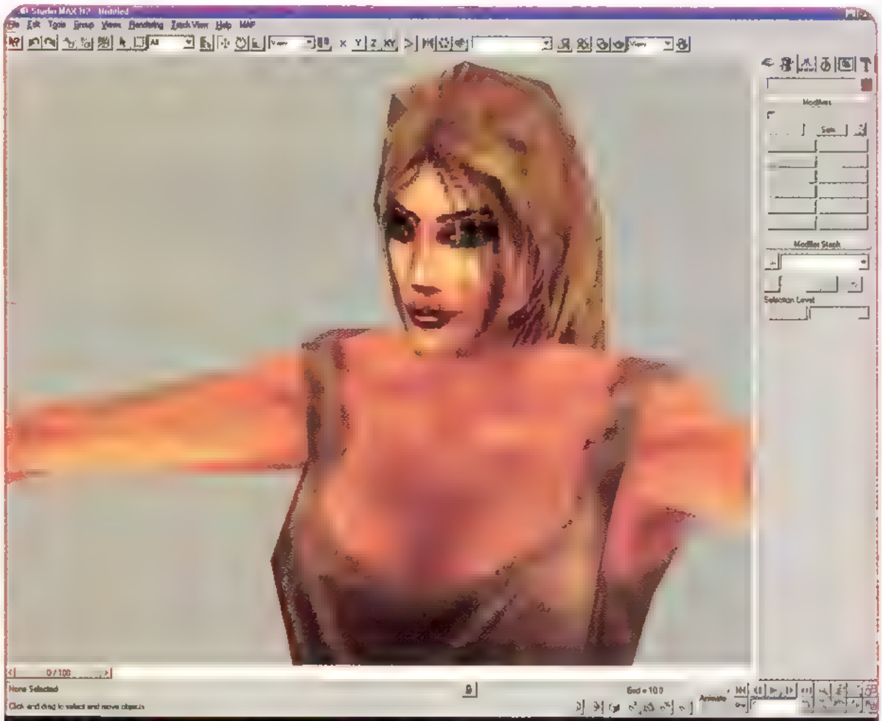

File Edit Tools Group Views Rendering Track View Help MAX
Modifiers
Modifier Stack
Selection Level
0 / 100
None Selected
Click and drag to select and move objects
End = 100
Animate

FIGURE 5. *Character before the addition of color vertices in 3D Studio MAX 2.5.*

Maximum Audio. Minimum Hassle.
www.qsound.com
PSYGNOSIS
SINGLETRAC
LOOKING GLASS STUDIOS
MICROPROSE
SOUND SOURCE INTERACTIVE
ACTIVISION
CAPCOM
SIMIS
THE DIGITAL VILLAGE
INFOGRAMES
GLASS GHOST
QSOUND
Come play with us.

Download a FREE Audio SDK at www.qsound.com/gamedev. Email us at dev.info@qsound.com or call 403-291-2492

lighting scheme is a blend of both high and low-saturation light sources, blue/yellow, purple/orange, purple/green, and blue/red combinations are all pretty safe.

Procedural coloring routines, such as the one in this example, are fine for making macroscopic adjustments to terrain and environments, but what if you want to do some fine detail work, such as coloring the vertices on a character?

Hand-Adjusted Color

Generating color vertices based upon lighting for characters is usually not effective. Characters traverse several lighting schemes while interacting with their environments, so it's virtually impossible to find a single set of color vertices that works for all cases. We can, however, modify the color vertices by hand to give highly detailed results. Coming up with ways to distinguish between damaged and healthy characters without blowing a texture budget can drive a good texture artist off the deep end. In this example we'll look at ways to accomplish the same effect using 3D Studio MAX's vertex coloring tool without adding a single pixel to the texture budget.

3D Studio MAX's Assign Vertex Color utility works in much the same way as Softimage's RenderVertexColors. Vertices receive their color data either from a texture map or from lights in the scene. We can also modify the vertex colors by selecting the vertices and assigning the colors directly. Figures 5 and 6 show a player character in a "before" and "after" mugshot. To

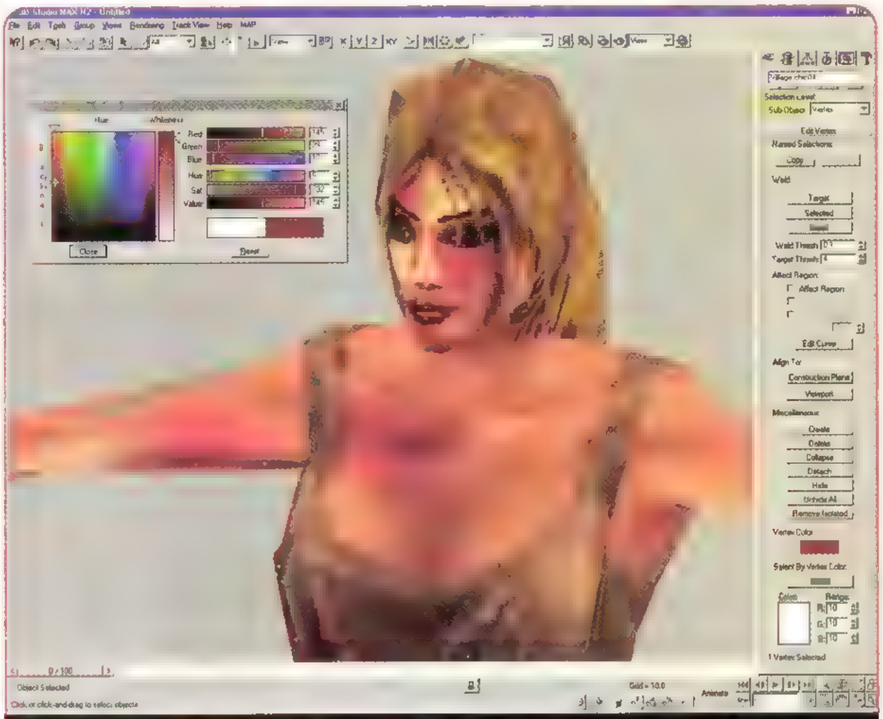


FIGURE 6. Hand-painted color vertices using MAX's Assign Vertex Color utility.

FIGURE 7. The two textures that create the terrain in Figure 8.

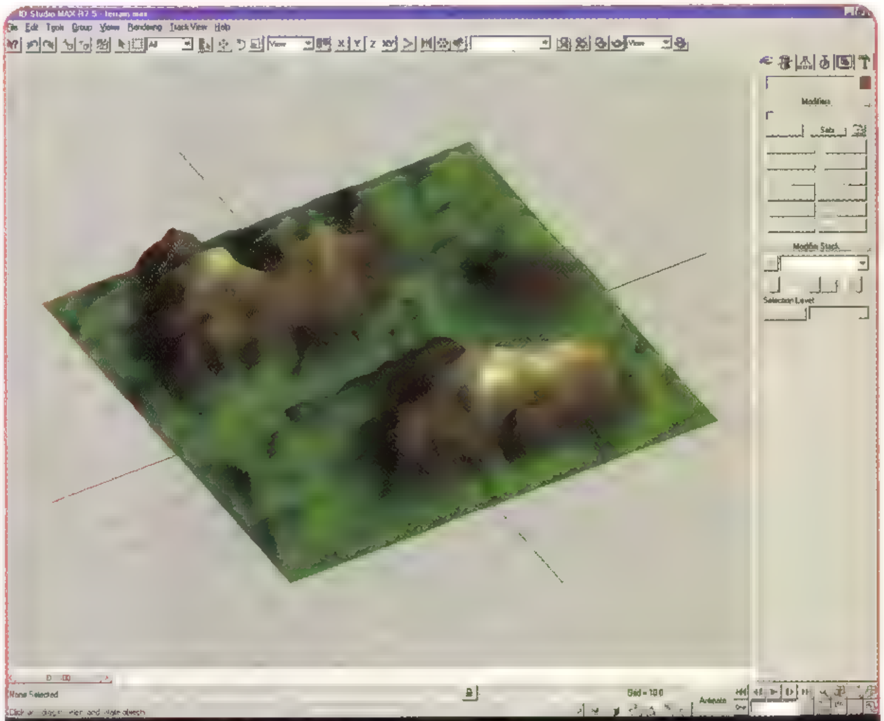


FIGURE 8. Textured terrain using color vertices.

get this same effect using textures, we would need to duplicate the entire set of textures for the face and upper body. Furthermore, until we get a really good 3D paint tool, the process of painting blood onto a character is anything but an exact science; you'll end up swapping back and forth between paint programs and 3D programs to get it right.

To achieve this effect using vertex colors literally took about five minutes. Using MAX's vertex selection tool, you simply select the vertices you want to color, and then select the color you want. The process is fully interactive, and the result updates in the shaded window as you change the colors. This same method can be used on static objects to create the illusion of dirt and grime, or to make an object appear damaged. The only limitation is the number of vertices in the model, because the resolution of the color vertices is directly linked to the detail you can achieve.

SUGGESTIONS FOR IMPLEMENTATION. Most engines allow you to store multiple versions of each object, usually for level of detail. One way to take advantage of this process is to store multiple versions with identical vertex counts, but with differing color vertex sets. And because the storage space required for 3D geometry is virtually nothing compared to that required for texture space, it's possible to get a large degree of diversity at virtually no expense. The last example uses both of the previous techniques, as well as a third aspect of the color vertex process: baking a texture map into the geometry.

Large-Scale Terrains

One of the problems encountered by people doing flight sims is how to texture map extremely large expanses of terrain. This is one case where the application and use of color vertices provides an interesting solution to an otherwise daunting problem.

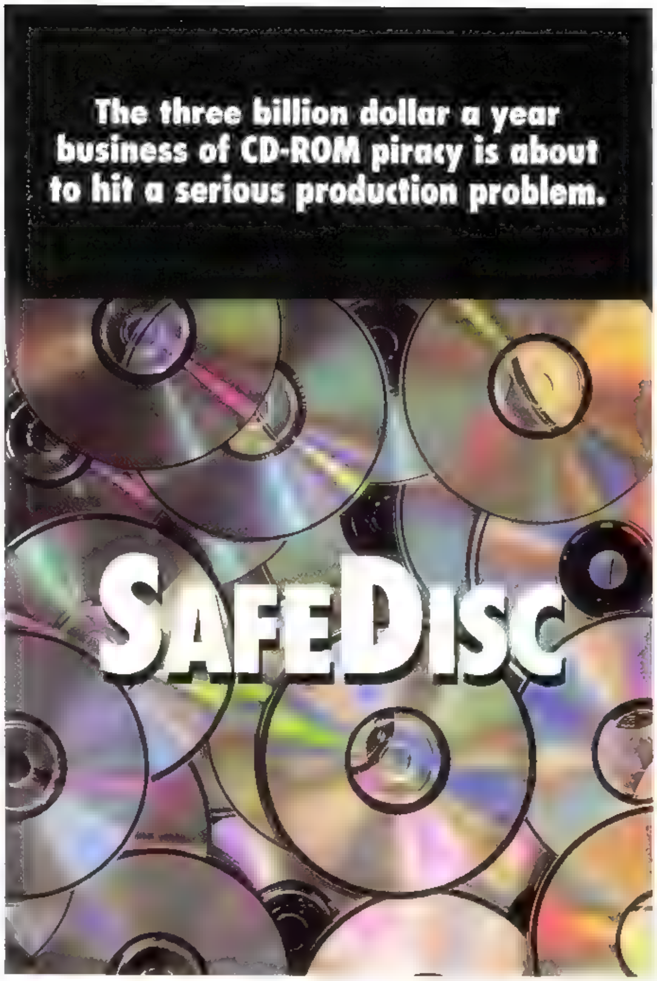
Figure 7 shows the two textures that were used to create this terrain. The first is a simple color gradient and the second is a tiling noise map. The gradient texture is applied with a horizontal projection, so that vertices of a given height all correspond to the same color. Note the high, uniform vertex density in this example — this would probably represent a fairly highly detailed patch.

Figure 8 shows the fully lit terrain after the application of color vertices, and the small noise map which serves to provide the requisite pixel detail. Again, note that there are no lights in this scene; the lighting information has already been stored in the color vertices. The final result is pretty impressive, since we only take up around 20K of texture space, and we get several square miles of detailed terrain in the bargain.

SUGGESTIONS FOR IMPLEMENTATION. Try a combination of all three processes; bake the lighting and texture values in, and then go back and adjust areas of interest by hand. Try using a high-density mesh with a top-down project of a photo-realistic texture map, then go back with a noise map and tweak the result.

We've just scratched the surface when it comes to ways to effectively use color vertices, and with our polygon-budgets continually on the rise, we are sure to see even more.

Special thanks to Melisa Bell, Hugh Jamieson, Hayley Reed, and Louise Smith. ■


The three billion dollar a year business of CD-ROM piracy is about to hit a serious production problem.
SAFEDISC

Consumer copiers, hackers, and commercial pirates cost you and the PC entertainment, education, and reference CD-ROM publishing industry more than $3 billion a year Which is exactly why you need SafeDisc.

SafeDisc prevents CD-Recordable drives and other remastering devices from producing useable copies of original CD-ROMs. It does not interfere with game or program playability . its disc authentication technology assures that only original discs are playable . . . and SafeDisc is available to all mastering houses and replicators. SafeDisc is from Macrovision Corporation, the leading copy protection technology company in the world. Macrovision offers its security technologies in over 35 countries on six continents. In fact, our video copy protection technology has been the de facto standard for protecting Hollywood's home video movies for over a decade Now, for the first time, a truly effective security technology is being made available to the multimedia software market from an industry leader. So give us a call at +1 (408) 743-8600 Or visit our website at www.macrovision.com. Then find out more about SafeDisc. And start bringing all that lost revenue right back to your bottom line–where it belongs

MACROVISION
Protecting your image

GROUP DISCOUNTS AVAILABLE!

Order some fresh, tasty game development classes for your entire team.

The 1998 GDC RoadTrips!

SEATTLE: November 9 & 10
AUSTIN: November 16 & 17
S.SAN FRANCISCO: November 21 & 22
BALTIMORE: December 10 & 11
BOSTON: December 12 & 13

The Game Developers Conference is serving up some spicy learning/networking events this year, right to your doorstep. In 1-2 concise days, you'll hear what the hottest designers are thinking about as they build their latest games, and debate the merits of next generation technology with developers who know their stuff. Fill yourself up with classes on **game design**, **programming**, **visual arts**, **production**, **business/legal**, and **audio**. New this year—a preconference day exploring one burning topic in depth. Bring your appetite for new approaches, tools, and business contacts!

(No extra charge for hometown delivery)

GAME Developers CONFERENCE ROADTRIPS

Register today for a GDC RoadTrip.

Also by the Miller Freeman Game Group:
Game Developer
Gamasutra

Go To www.gdconf.com/gdmg
Call 415-905-2388
E-mail cgdc@mfi.com

Trends in the Sales Channel

There are many changes happening in software sales channels today. As a result, the opportunities for game publishers and developers are shifting. This is especially true for game companies in the PC sector; the large chains and the big warehouse clubs are eating up the market.

The value of PC game sales through the various retail channels comes to about $3 billion. Let's look at how these 4,200 titles find their way into homes.

Skewed SKUs

According to PC Data, software and computer stores carry approximately 1,500 SKUs each (these include all software — games, applications, utilities, and so on). Direct mail vendors carry nearly 700 SKUs, and the Wal-Mart-types carry about 500 software titles. (Note that CompUSA, with over twenty thousand square feet of store space, carries fewer than 6,000 SKUs.)

Compare these numbers to a large bookstore like Barnes & Noble, which can carry up to 12,000 book titles at any one time. Granted, there may only be one or two copies of some books in stock, but most titles can sit there on the shelf quite happily for six months or more. Unfortunately for us, these conditions don't apply to software. Software retail establishments simply don't have enough shelf space for all of the titles the game industry puts out, and this shortage of "shelf space" is especially detrimental to independent developers and self-publishers. (Although, sadly, since the top ten game titles account for 15 percent of all the game industry's revenue, the lack of shelf space might be a moot point.)

Internet game sales aren't even a blip on the radar right now. Many publishers, however, feel that the Internet could account for as much as 20 percent of all revenues within the next five years. Could this be possible? Even if this prediction pans out, I don't think that we will see a jump in the number of SKUs retailers put before consumers. The reason is that Internet vendors, like direct mail vendors, don't want to stock a large selection of titles. These firms want to pitch a product, get the order, collect the money, and ship it from a consignment warehouse belonging to a distributor or publisher. To get 4,200 SKUs out before the public would require one monster consignment warehouse, and I doubt that level of cooperation between the various publishers and distributors will happen anytime soon.

What about taking the Internet to the next step and making it a delivery vehicle for software as well? Ideally, consumers would embrace the idea of purchasing and downloading a title, thereby eliminating the box and printed documentation. But at today's retail prices that's a no-go. Psychologically, when you pay big money for goods you want something more substantial than a three-hour download. Hence, I'm not enthusiastic about the Internet's chances as a fulfillment mechanism.

Segment and Conquer

I've noticed an interesting phenomenon in the retail sales channel: the best place to sell game titles is not necessarily the best place to buy game titles. For example, although a computer store like CompUSA or Fry's Electronics may continue to be the primary source of sales for PC games by market share (see Table 1), these types of stores don't fulfill the entire game industry's needs. The reason is that computer stores cater to *all* computer users, from novice consumers all the way up to large corporate buyers. With a desire to move volume hardware, peripherals and software (and the commensurate margin squeeze that accompanies this kind of selling), the computer superstores want publishers to prominently display their titles and offer them at low prices to boot. This is how Broderbund keeps selling copies of MYST and Print Shop Deluxe: the titles are easy to recognize, easy to find, and a nice item to dunk in the shopping cart as a person walks down the aisles. Unfortunately, not every publisher can afford lavish displays or cut margins enough to succeed in computer superstores.

Next, let's examine consumer electronics stores. These stores, such as Best Buy and Circuit City, currently sell tremendous amounts of software. Recently, though, they have begun to cut back on their computer products sections to enlarge their consumer electronics and appliances sections (which have fatter margins and are poised for high growth with the emergence of digital television products). So I don't view stores like Best Buy as a source of growth for the PC game industry.

Toy stores such as Toys 'R Us can't help the PC game industry much. The young demographic of the Toys 'R Us shopper certainly appeals to publishers of console games, but not necessarily to PC games publishers.

A Console Future?

I'm more optimistic about console games. Console game publishers are not too reliant upon the traditional software retail channels like CompUSA, and

Omid Rahmat works for Doodah Marketing as a copywriter, consultant, tea boy, and sole employee. He also writes regularly on the computer graphics and entertainment markets for online and print publications. Contact him at omid@compuserve.com.

TABLE 1. *Top retailers by software sales alone (Source: Computer Retail Week).*

Store Type	Name	Number of Stores	Total Software Sales ($M)
Consumer Electronics Store	Best Buy	285	$3,422.00
Computer Store	CompUSA	153	$3,178.00
Office Superstores	Office Depot	565	$2,350.00
Consumer Electronics Store	Circuit City Stores	493	$1,810.00
Computer Store	Computer City	96	$1,375.00
Office Superstores	Staples	523	$1,250.00
Office Superstores	OfficeMax	713	$1,080.00
Computer Store	Micro Center	13	$1,030.00
Department Store	Sears, Roebuck	835	$890.00
Warehouse Clubs	Sam's Club	444	$728.00
Computer Store	Fry's Electronics	16	$705.00
Mass Merchants	Wal-Mart Stores	1,700	$700.00
Warehouse Clubs	Costco	204	$502.00
Consumer Electronics Store	RadioShack	6,906	$341.00
Military Store	Army Air Force Exchange	142	$305.00
Software Stores	Egghead Computer	86	$280.00
Computer Store	PC Warehouse	82	$263.00
Software Stores	Electronics Boutique	470	$229.20
Computer Store	J&R Computer World	1	$195.00
Consumer Electronics Store	Future Shop	27	$194.80
Software Stores	Babbage's Etc.	466	$175.00
Consumer Electronics Store	Sun TV & Appliances	48	$160.00
Consumer Electronics Store	The Good Guys!	76	$140.00
Retail Dealer	Computown	5	$140.00
Retail Dealer	Creative Computers	8	$126.00
Consumer Electronics Store	Nobody Beats the Wiz	49	$125.00
Retail Dealer	Computer Renaissance	187	$107.00
Bookstore	Barnes & Noble	1,013	$105.00
Consumer Electronics Store	Nationwide Computers	4	$95.00
Computer Store	CDW Computer Centers	2	$84.00
Toy Store	Toys R Us	686	$75.00
Retail Dealer	PC Club	15	$73.00
Warehouse Clubs	BJ's Wholesale	87	$70.00
Mass Merchants	Target Stores	799	$69.00
Computer Store	RCS Computer Experience	2	$59.00
Music Stores	Musicland	226	$58.00
Consumer Electronics Store	P.C. Richard & Son	40	$57.80
Consumer Electronics Store	American TV	8	$56.40
Retail Dealer	SBI Computer Warehouse	6	$56.00
Retail Dealer	Computer Ware	10	$52.00
Computer Store	DataVision	1	$50.00
Retail Dealer	Lucky Computers	18	$48.00
Retail Dealer	Computer Town	7	$46.00
Direct Mail	Global DirectMail	2	$25.00
Direct Mail	Micro Warehouse	NA	NA
Direct Mail	Insight Enterprises	NA	NA
Direct Mail	PC & Mac Connection	NA	NA
Direct Mail	Multiple Zones	NA	NA
Direct Mail	Damark	NA	NA

with Sega, Sony, and Nintendo spending hundreds of millions of dollars on Christmas promotions and sales, the console industry is set for healthy growth for the next two years.

The onslaught of console marketing dollars means that direct channels, such as mail order and Internet sales, have the most potential for the PC game industry. The factor damping the growth of these direct channels is simply the present industry structure. The top ten PC game publishers are loathe to damage their existing distribution channels, limited or not, by putting too much emphasis on direct sales. So the two-tier distribution model (where a big distributor warehouses products, takes orders directly from retailers, and ships these stores their inventory) will not see any significant changes for some time.

[illegible]

What we *can* expect from the top-ten PC game publishers is that they will squeeze more out of the existing distribution channels. These big publishers have already started to filter the titles carried by outlets such as CompUSA and Wal-Mart. These retailers see the game market as beneficial to their business, but they will become increasingly stringent about what products make it to their shelves. Just look at the way EA and GT Interactive already control their channels. The publishing community is becoming a closed set, a private club with a waiting list.

The only option available to small publishers and developers going the self-publishing route is to circumvent the whole process – to market directly to the consumers. It's analogous to the situation facing the makers of PC clones. These firms must market directly via mail order and the Internet, too. Some clone vendors may someday grow up to be the next Dell or Gateway, but for most that's a long way off.

PC game developers have to start viewing publishers as brand equity in the channel. If developers want independence, they must look for niche audiences that can be targeted with direct sales. The losers in this scenario will be the small retailers who can't compete with the superstores or take a slice of the direct marketing business.

I believe that PC game development is going to shrink in the next two years, and be replaced in part by console development projects. The cost of goods and sales on games will rise, while the retail prices will continue to drop. These big publishers with large catalogues of titles and economies of scale will be the firms that flourish.

[illegible]

1. Cendant
2. Broderbund
3. The Learning Company
4. EA
5. GT Interactive
6. LucasArts Entertainment
7. Microprose
8. Interplay
9. Activision
10. Mindscape

The Most Complete Solution for Professional 3D Data and Animation Translation!

PolyTrans™ v2

With 3DS MAX* Plugin Support!

New!

Release 2.1 Now Supports Native 3D Studio MAX Plugins, OpenFlight®, Trimmed NURBS Surfaces, Rhino* (OBJ), Full IGES 5.3, 3DS/Lightwave*/DirectX Animation Conversion, Renderware*, ACIS* SAT (solid models) and more!*

Download Demos From http://www.okino.com

3D Studio®
Apple® 3DMF®
Alias® triangle
Detailer®
DirectX®
DXF®
IGES 5.3
Imagine®
Lightscape®
Lightwave®
OpenFlight®
OpenGL C Code
POV 2 & 3

Windows NT/95/3.1 & Silicon Graphics

Pro/E® slp
Renderman® RIB
Renderware®
Softimage® hrc/dsc
SAT (ACIS®)
Solid Edge® Part
STL
Strata® StudioPro
trueSpace® 2 & 3
USGS DEM
VistaPro® DEM
VRML™ 1&2
Wavefront® (+NURBS

"PolyTrans is '*the*' premiere 3D conversion tool available today."
-Peter-Josesph Kovach, 3D Book Author & VC++ Dev. Journal Magazine Writer

"PolyTrans is by far the most versatile (translator) with the most reliable results."
-Joe Morley, Sales & Marketing Director, REM Infografica (3D Models Bank)

"PolyTrans stands alone among 3D file format converters currently on the market."
Brendon Perkins, 3D Design Magazine, May 1997

"PolyTrans is the only tool we have found that can correctly convert our 3D databases."
-Ray Arell, Architecture Validation Engineer, Intel Corp

Okino
Computer Graphics

Toll Free: 1-888-3D-OKINO, WEB http://www.okino.com
T (905) 672 9328, F 672-2706, Email sales@okino.com
[illegible] and PolyTrans are registered trademarks, and Okino is a trademark of [illegible]

3D
The Architects of Cyberspace
3D MODELING
SCULPTING
DIGITIZING
CYBERPROPS
1(800) 993-4621
(310) 305-1981
www.3name3d.com
info@ywd.com

run

time Mip-Map filtering

by Andrew Flavell

Graphics programmers are constantly looking for ways to improve the realism of the graphics in games. One of the simplest techniques employed to do this is texture mapping, and while texture mapping does add considerable realism to a scene, it also adds a number of new problems. The most obvious visual problems that appear when using textures in a scene are the aliasing artifacts that are visible when texture-mapped polygons are some distance from the viewpoint. If you're moving rapidly around your virtual world, these artifacts appear as flashing or sparkling on the surface of the texture. Or, if the viewpoint is fixed, the artifacts appear as unwanted patterns within the texture after it has been mapped to a polygon. This is clearly visible in Figure 1, where the checkered texture map becomes distorted as its distance from the viewpoint increases.

Andrew Flavell is yet another out-of-work PhD grad, wondering why he spent all of those years at school studying graph-theory and Markov models. Questions regarding the article and job offers can be sent to mipmapping@weta3d.com

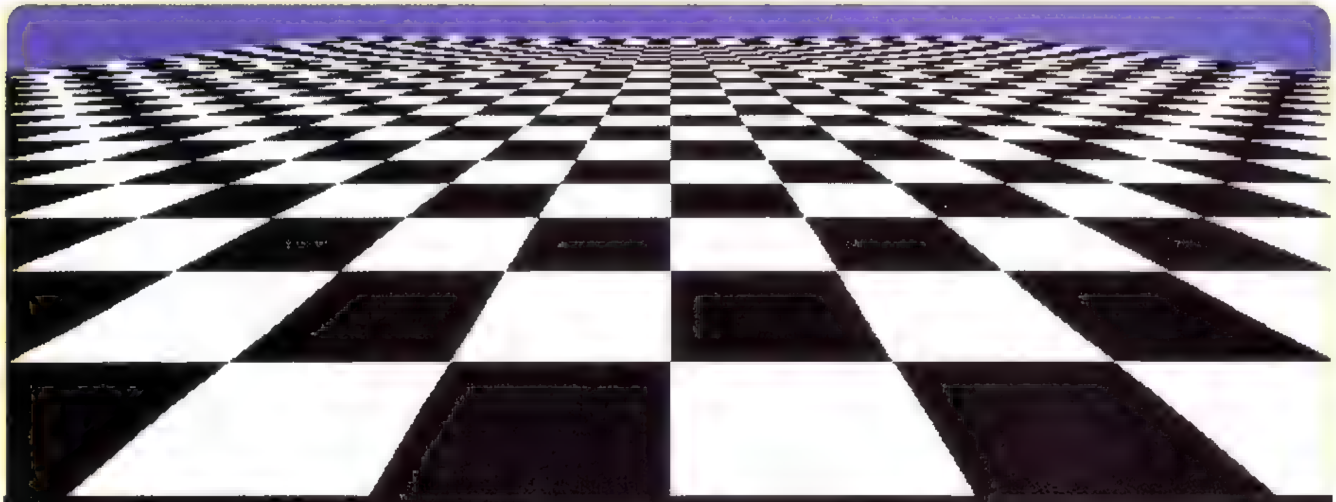


FIGURE 1. *Checkerboard with no MIP-mapping.*

MIP-mapping helps alleviate this problem. The acronym MIP comes from the Latin phrase *multum in parvo*, meaning "many things in a small place." Researchers at the New York Institute of Technology adopted this term in 1979 to describe a technique whereby many pixels are filtered and compressed into a small place, known as the MIP-map. To see how MIP-maps improve visual clarity, see Figure 2, in which MIP-mapping with bilinear filtering has been used to smooth the texture.

In order to understand what is what's causing the problems in the Figure 1, you have to look within the texture-mapping renderer and understand how the process of sampling the texture maps affects what's displayed on the screen. Look at Figure 3A, in which a sine wave is being sampled at a much higher frequency than the wave itself. As you can see, a fairly good representation of the wave can be obtained from these samples. However, if the sampling frequency drops to exactly two times the frequency of the wave, as shown if Figure 3B, then it's possible that the sampling points will coincide with the zero crossing points of the sine wave, resulting in no information recovery. Sampling frequencies of less than twice that of the sine wave being sampled, as shown in Figure 3C, causes the information within the samples to appear as a sine wave of lower frequency than the original. From these figures, we can guess that for complete recovery of a sampled signal, the sampling frequency must be at least twice that of the signal being sampled. This is known as the Nyquist limit. So, from where does the seemingly magic value of twice the signal being sampled come? In order to answer, that we'll have to digress a bit further and take a stroll into the Fourier domain.

[illegible]

A complete discussion of Fourier theory could take up several books by itself, so for those of you who haven't suffered through a signal-processing course at college, I suggest that you take a look at the text by Bracewell that's mentioned at the end of this article. What follows is a very limited introduction to Fourier transforms and sampling, but it should be enough to demonstrate how the Nyquist limit is derived.

Figure 4A shows a plot of the function $h(t)=\mathrm{sinc}^2 x$ and a plot of its Fourier transform, $H(f)$. It's convenient to think of $H(f)$ as being in the Fourier (or frequency) domain and of $h(t)$ as being in the time domain. (If you're wondering why I

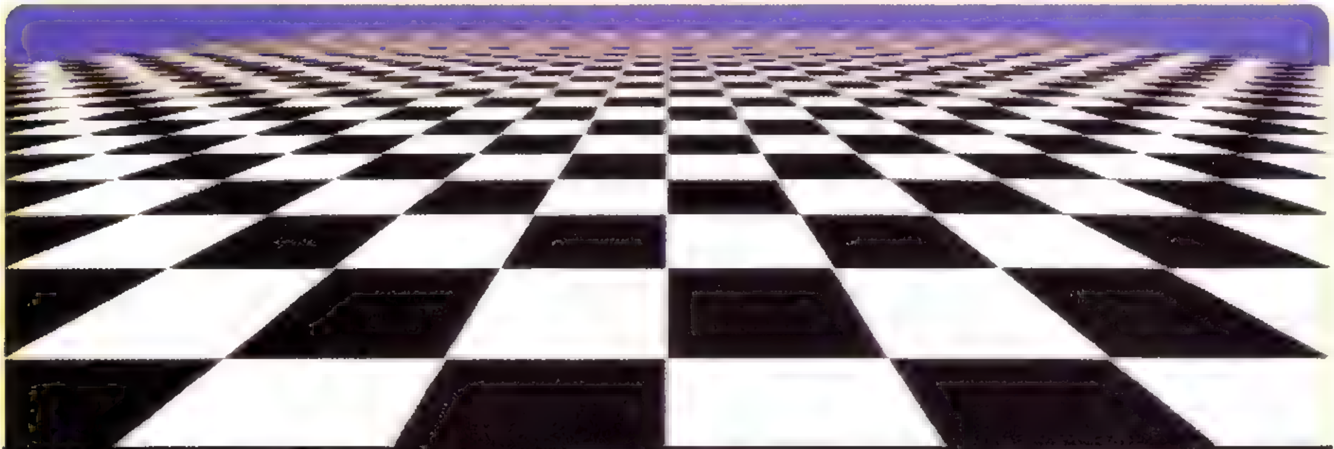


FIGURE 2. *Checkerboard with MIP-mapping and bilinear filtering.*

chose to use $\mathrm{sinc}^2 x$ for this example, it's because it has a simple plot in the frequency domain.) To convert from the time domain to the frequency domain, the following transform is applied to $h(t)$:

$$H(f)=\int_{-\infty}^{\infty} h(t)e^{-i2\pi ft}dt$$

Eq. 1

In this form of the Fourier transform, f defines the frequencies of the sine waves making up the signal, and $i=\sqrt{-1}$ tells us that the exponential term is complex (that is, it has both real and imaginary parts). The operator $\supset$ is often used to denote "has the Fourier transform," so we can write $h(t)\supset H(f)$. Figure 4B shows the train of impulses used for sampling and the Fourier transform of the impulses. An impulse, denoted as $\delta(t)$, is a theoretical signal that is infinitely brief, infinitely powerful and has an area of one. An interesting property of an impulse train with a period of T_s is that its Fourier transform is an impulse train with a period of $1/T_s$.

$$\sum_{n=-\infty}^{\infty}\delta(t-nT_s)\supset\frac{1}{T_s}\sum\delta\left(\frac{f-n}{T_s}\right)$$

Eq. 2

The effect of sampling $h(t)$ with the sampling function $s(t)$ is shown in Figure 4C. In the time domain, the sampling can be thought of as multiplying $h(t)$ by $s(t)$, and in the frequency domain, it can be thought of as the convolution of $H(f)$ and $S(f)$.

$$h(t)s(t)\supset H(f)*S(f)$$

Eq. 3

Convolution of any two functions $f(x)$ and $g(x)$ is given by

$$f(x)*g(x)=\int f(u)g(x-u)du$$

Eq. 4

If the thought of plugging the Fourier transforms of both $h(t)$ and $s(t)$ into Equation 4 has you wanting to skip to the current Soapbox article (p.72), just hold on a second — it isn't as bad as it looks. The convolution of a single impulse located at $t=t_0$, with $h(t)$ is just the value of $h(t)$ shifted to that location.

$$h(t)*\delta(t-t_0)=\int h(u)\delta(t-t_0-u)du=h(t-t_0)$$

Eq. 5

We can apply the result of Equation 5 to find the convolution of $H(f)$ and $S(f)$.

$$h(t)s(t)\supset\frac{1}{T_s}\sum H\left(\frac{f-n}{T_s}\right)$$

Eq. 6

Equation 6 simply means that the result of the convolution of $H(f)$ and $S(f)$ is such that $H(f)$ is duplicated at intervals of $1/T_s$, as can be seen in Figure 4C. The $\mathrm{sinc}^2 x$ function is bandlimited (that is, its bandwidth is limited) to f_{max}, so the only requirement needed to ensure that there are no overlapping portions in the spectrum of the sampled signal is that $f_s>2f_{max}$, where $f_s=1/T_s$. So, this is from where the Nyquist limit comes. As you can see in Figure 4D, if the sampling frequency drops below $2f_{max}$, adjacent spectra overlap at higher frequencies, and these frequencies are then lost in the resulting signal. However, instead of disappearing completely, these high-frequency signals reappear at lower frequencies as aliases; this is where the term aliasing originated. To prevent aliasing from occurring, either the signal being sampled must be bandlimited to less than $2f_s$ or the sampling frequency must be set to be higher than $2f_{max}$

MIP Mapping Basics

Let's look at how MIP-mapping helps to reduce aliasing artifacts in our texture-mapped image. Remember that texture mapping is designed to increase the realism and detail in scenes. However, all of the fine details in the texture maps are effectively-high frequency components and they are the cause of our aliasing problems. Since we can't really modify our sampling frequency ($1/\Delta U$ and $1/\Delta V$ in the texture-mapping portion of our renderer), we have to filter the textures to remove the high-frequency details.

Although it would be possible to filter each individual texel at run time, this would require a significant amount of effort. To get around this problem, we can use MIP-maps, which are made up of a series of prefiltered and prescaled textures. The filtering of the textures either can be carried out during the startup of your game, or you can

FIGURE 3. *Sampling a sine wave with varying sampling intervals.*

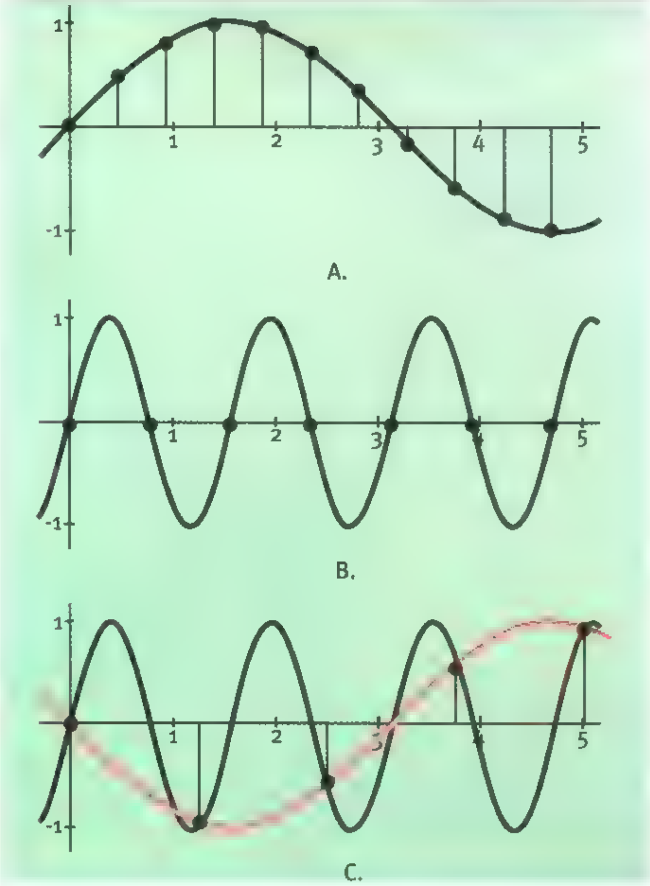
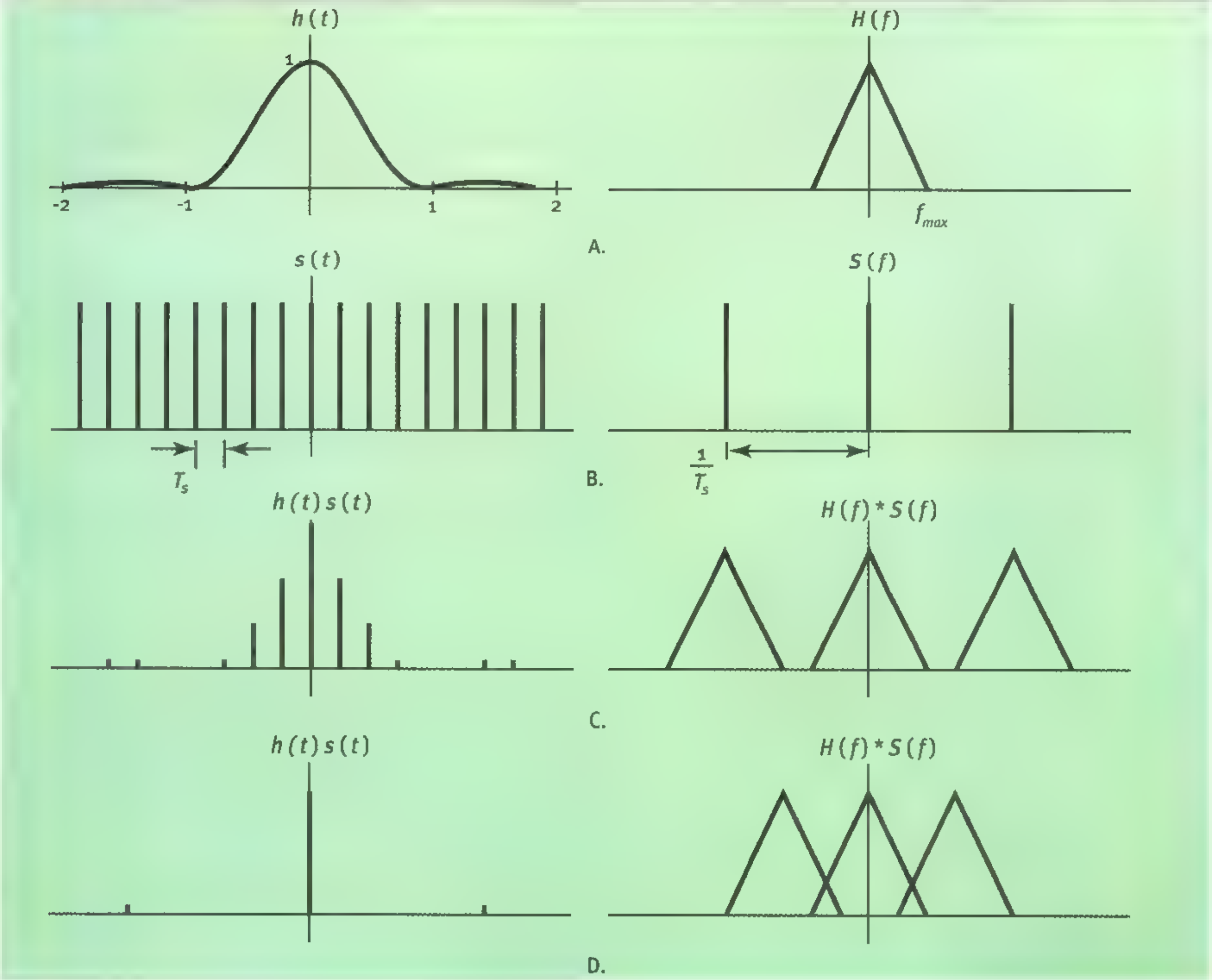


FIGURE 4. *Fourier analysis of sampling.*



prefilter all of your textures during development. Another alternative with some graphics cards, such as those using the Nvidia RIVA 128 accelerator, is to have the card automatically generate MIP-maps for you when textures are uploaded into video memory. Figure 5 illustrates the pyramid-like structure formed by the MIP-map for a 64×64 pixel texture. As you can see in the figure, the level of detail (LOD) decreases as the MIP-map level increases. Once textures have been filtered, all you have to do at run time to achieve basic per-polygon MIP-mapping is to select the correct MIP-map level (or LOD) for the desired texture and pass this to the renderer.

Generating MIP Maps

There are a number of ways to generate MIP-maps. One option is simply to prebuild them using a graphics processing tool such as Photoshop. The alternative is to generate your MIP-maps on the fly. Prebuilding MIP-maps requires about 30 percent more storage space for your textures when you ship

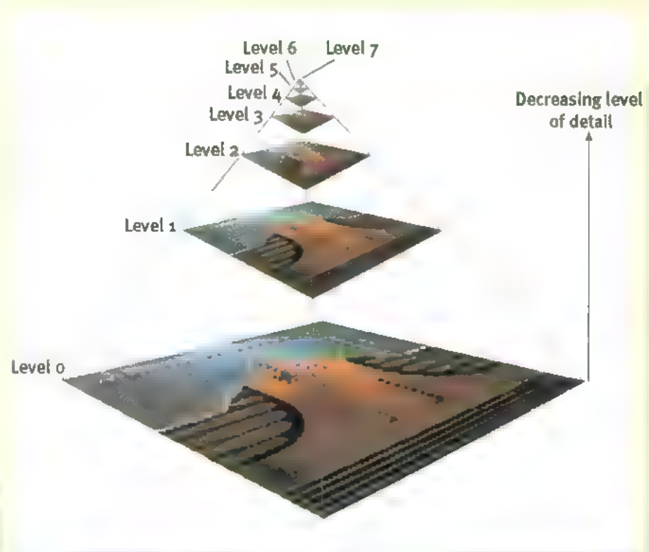
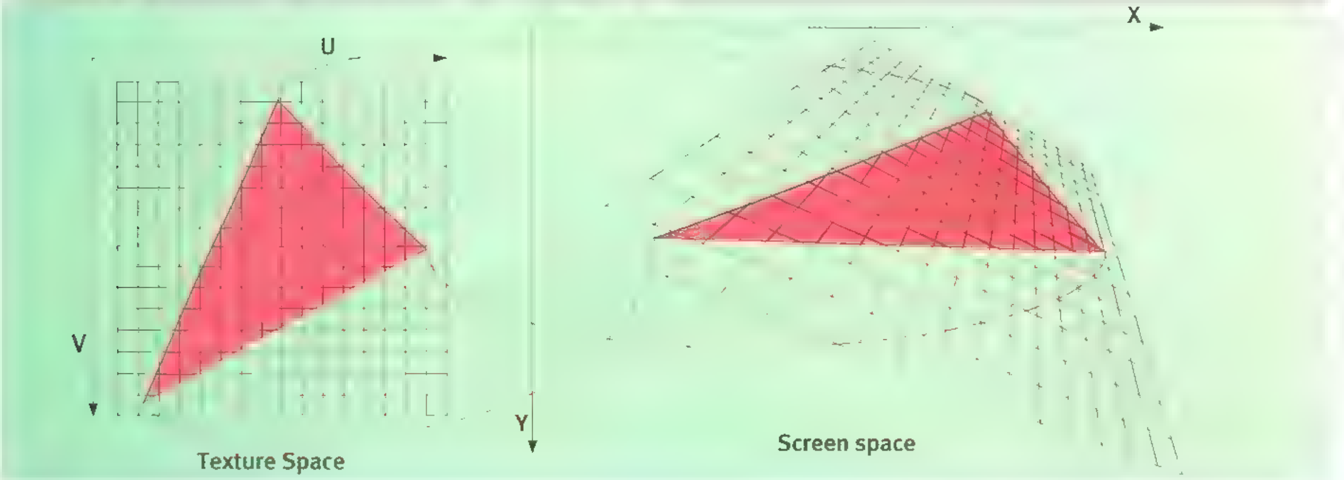


FIGURE 5. *A MIP-map pyramid.*

FIGURE 6. *Texture distortion after perspective projection.*



your game, but it gives you finer control over their generation and it lets you add effects and details to different MIP levels. Regardless of which method you choose, MIP-maps require 30 percent more storage space at run-time than the original textures, so they can have a significant effect on your game's memory requirements.

Let's begin by generating a MIP-map for an 8-bit texture. Generating a MIP-map is a fairly simple process and although there are many possible filtering techniques that could be applied during MIP-map generation, a standard box filter usually suffices. The first level of the MIP-map is generated by taking the raw input texture and copying it directly into the MIP-map data structure shown in Listing 1. [In the interest of conserving editorial space, code listings are available for download from *Game Developer*'s web site. -Ed.]

Creating the rest of the MIP levels of a texture is an iterative process. Each successive level of the MIP-map is generated using the preceding, larger, MIP-map level. As each level of the MIP map is created, it's stored consecutively in memory, and a pointer to the starting memory address of the MIP level is stored as well, so that the game engine can quickly access the correct LOD during rendering.

The first step in generating a new pixel value is to calculate the average color value from the four corresponding pixels in the preceding level, as shown in Listing 2. As there is a palette associated with the texture in this example, once the new color value has been calculated, we need to search the palette associated with this texture to find the entry that most closely matches the desired color. This process is shown in Listing 3. The color search process is quite simple, but it can be time-consuming, as we need to search the palette for a color match for every pixel in each level of the MIP-map. Thankfully, this step is only required during the initialization of the MIP-map, so it's not much of a problem. However, if you want to perform other effects during rendering (such as bilinear or trilinear filtering), the search process will be too slow.

In this case, we'll need to use 16- or 24-bit textures. Because most graphics cards currently support 16-bit screen depths, we'll use 16-bit textures here. The process of building MIP-maps for 16-bit textures is very similar to that used for 8-bit textures, as you can see in Listing 4. Because 16-bit textures don't require a palette, averaging the color values from the four corresponding pixels in the preceding level directly gives each new pixel value. One problem that can occur as a result of repeatedly averaging the color values for each LOD is that the texture map will become darker at each successive LOD. You can compensate for this effect by adding a small amount to each color component at each LOD, but this compensation usually isn't necessary, as the loss of color during the entire process is very small.

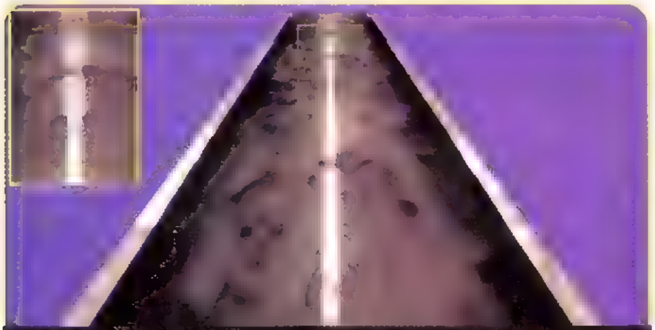


FIGURE 7. *Texture mapping with incorrect LOD.*

Applying MIP maps at Run Time

Figure 6 shows some of the problems you can encounter when selecting which LOD to apply at run time. In the figure, the rectangular texture that's mapped onto the triangle in texture space is transformed into a quadrilateral in screen space, and the perspective projection of the texture causes the

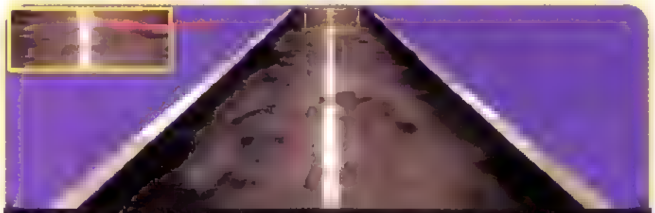


FIGURE 8. *Road rendered with per-polygon MIP-mapping.*

individual texels to become quadrilaterals of varying sizes. In a case such as this, where the orientation of a polygon is skewed in screen space, determining the best LOD to apply to a polygon is especially crucial if you want to produce good visual results. If the chosen LOD is too high (the texture dimensions are too large), aliasing will occur in the texture. If the LOD is too low (the dimensions of the texture are too small), then the image will appear blurred. For example, the LOD chosen for the texture in Figure 7 is much too low, as can be seen by the large texels visible in the inset zoomed image. Many different methods can be used for LOD selection, all of which have advantages and disadvantages. The two well-known methods that we'll examine here are the selection of the LOD based on the area of the texture in screen space, and the selection of the LOD using the projected *u* and *v* vectors.

One further point to consider here is that it's possible that a different number of texels map to each pixel in screen space. As a result, correct LOD selection requires calulating the LOD for each pixel. Calculating which LOD to use can be quite slow; consequently most software renderers (and quite a few older hardware accelerators) calculate the LOD on a per-polygon or per-triangle basis. An added advantage of per-polygon MIP-map selection, especially for software-based renderers, is that you can use smaller versions of textures for distant (smaller) polygons, helping to reduce the amount of processor cache that's required during texturing operations. However, per-pixel LOD selection lets you do a number of other things with MIP-mapping, including point sampling, bilinear filtering within a single LOD, or trilinear filtering between the two closest LODs.

Per Polygon MIP Mapping

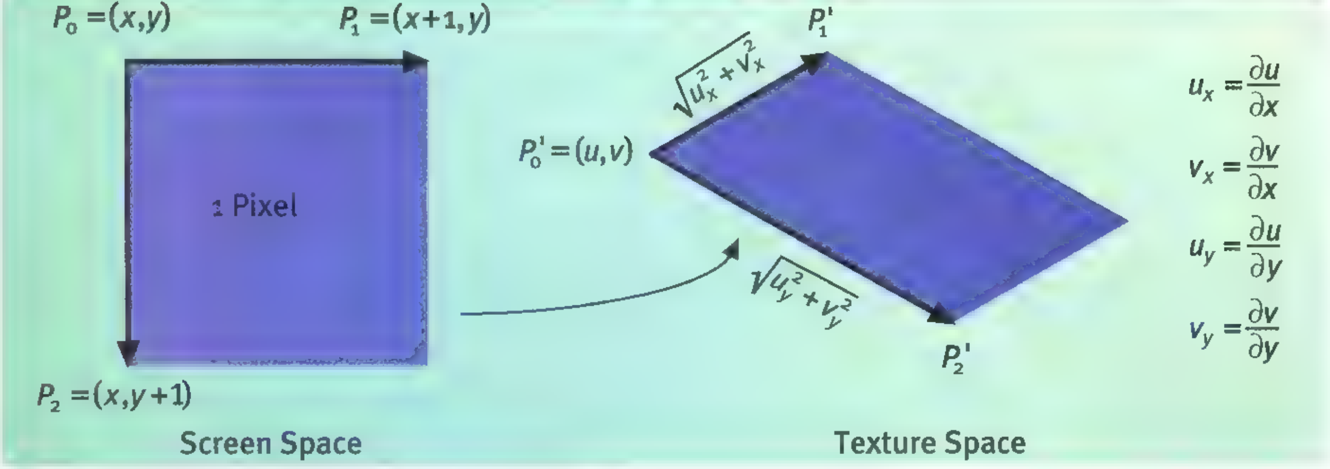
Per-polygon MIP-map selection is the least expensive method from a computational standpoint, becuse you only do MIP-map selection once per polygon. There are, however, a couple of drawbacks to this approach. One problem is that adjacent polygons that share a texture may be drawn using differing LODs; this will be appear as a discontinuity in the texture when displayed on the screen (this is called MIP-banding). Figure 8 shows a small amount of MIP-banding that is occurring due to the use of per-triangle MIP-mapping. Another problem is that visible popping may occur as a texture's LOD is changed due to movement of the viewpoint (or the polygon).

AREA-BASED LOD SELECTION. Area-based LOD selection complements per-polygon MIP-mapping techniques. In this method, you select the LOD by determining the amount of texture compression that will be applied to the texture in screen space. To determine the proper texture compression, you calculate the area of the polygon in screen space and the area, in texture space, of texture that is mapped onto the polygon. As shown in Listing 5, you can determine the ratio of texels to pixels and then determine which LOD to use. The *u* and *v* dimensions of each successive LOD are one-half the size of the preceding LOD, so each successive LOD has one-quarter the area of the preceding level. During LOD selection, we step up one level in the MIP-map pyramid for each multiple of four that the texel area is greater than the pixel area. For example, if the texel-to-pixel ratio is 3:1, we would select MIP-map level zero, or, if the texel-to-pixel ratio is 7:1, we would select MIP-map level one. Once the LOD has been selected, we can pass a pointer to the correct LOD, along with the LOD's dimensions, to our normal texture-mapping routines. One problem with any approach that uses the projected area of the polygon and the texture area as the basis for LOD selection is that aliasing will tend to occur whenever a projected polygon is very thin, due to the anisotropic nature of the texture compression (that is, the texture is compressed more in one dimension than the other).

[illegible]

Per-pixel MIP-mapping offers far better control of LOD selection than per-polygon MIP-mapping, and it also permits additional texture filtering — but at some additional cost. All of the per-pixel methods require storage of the entire MIP-mapped texture in memory, and adding LOD selection to the inner loop of a renderer's texture-mapping routine can significantly reduce rendering performance. Fortunately, most of today's 3D chips support per-pixel MIP-mapping with bilinear filtering (a few of the latest devices even support trilinear filtering), so we'll look at what it takes

FIGURE 9. A single pixel back-projected into texture space.


$P_0 = (x,y)$
$P_1 = (x+1,y)$
1 Pixel
$P_2 = (x,y+1)$
Screen Space
P_1'
$\sqrt{u_x^2 + v_x^2}$
$P_0' = (u,v)$
$\sqrt{u_y^2 + v_y^2}$
P_2'
Texture Space
$u_x = \frac{\partial u}{\partial x}$
$v_x = \frac{\partial v}{\partial x}$
$u_y = \frac{\partial u}{\partial y}$
$v_y = \frac{\partial v}{\partial y}$

to implement sophisticated per-pixel MIP-mapping. Although we could use area-based LOD selection here also (we'd need to calculate the texture area underneath each pixel rather than for the entire polygon), we'll look at an all-together more accurate method.

EDGE COMPRESSION-BASED LOD SELECTION. In 1983, Paul Heckbert probably examined more LOD calculation techniques than he'd care to remember before he decided that techniques based on the compression that a texture suffers along the edge of a pixel seem to work best. Figure 9 shows a single pixel in screen space and the corresponding parallelogram in texture space. To prevent aliasing from occurring, we want to select the LOD based on the maximum compression of an edge in texture space. This corresponds to the maximum length of a side in texture space, which is given by Equation 7.

$$\max\left(\sqrt{u_x^2 + v_x^2}, \sqrt{u_y^2 + v_y^2}\right)$$

Eq. 7

The values of u_x, u_y, v_x, and v_y are given by four partial derivatives. Because we already know how to calculate the *u* and *v* values for any pixel on the screen, we can use this knowledge to determine the partial derivatives. We know that, given the *u/z*, *v/z*, and 1/*z* gradients in *x* and *y*, and the starting *u/z*, *v/z*, and 1/*z* values at the screen origin, the *u* and *v* values for the texture at any pixel can be found using Equations 8 and 9. The notation in Equations 8 through 19 is derived from Chris Hecker's series on perspective texture mapping, which can be found on his web site (see "Acknowledgements" for the URL).

$$u = \frac{\mathrm{dUOverZdX} * x + \mathrm{dUOverZdY} * y + \mathrm{UOverZ}_0}{\mathrm{dOneOverZdX} * x + \mathrm{dOneOverZdY} * y + \mathrm{OneOverZ}_0} = \frac{\mathrm{UOverZ}}{\mathrm{Z}}$$

Eq. 8

$$v = \frac{\mathrm{dVOverZdX} * x + \mathrm{dVOverZdY} * y + \mathrm{VOverZ}_0}{\mathrm{dOneOverZdX} * x + \mathrm{dOneOverZdY} * y + \mathrm{OneOverZ}_0} = \frac{\mathrm{VOverZ}}{\mathrm{Z}}$$

Eq. 9

We can use these results to find the partial derivatives, as shown in Equations 10 through 13.

$$u_x = \frac{\mathrm{Z} * \mathrm{dUOverZdX} - \mathrm{UOverZ} * \mathrm{dOneOverZdX}}{\mathrm{Z}^2} = \frac{c + ay}{\mathrm{Z}^2}$$

Eq. 10

$$v_x = \frac{\mathrm{Z} * \mathrm{dVOverZdX} - \mathrm{VOverZ} * \mathrm{dOneOverZdX}}{\mathrm{Z}^2} = \frac{d + by}{\mathrm{Z}^2}$$

Eq. 11

$$u_y = \frac{\mathrm{Z} * \mathrm{dUOverZdY} - \mathrm{UOverZ} * \mathrm{dOneOverZdY}}{\mathrm{Z}^2} = \frac{e + ay}{\mathrm{Z}^2}$$

Eq. 12

$$v_y = \frac{\mathrm{Z} * \mathrm{dVOverZdY} - \mathrm{VOverZ} * \mathrm{dOneOverZdY}}{\mathrm{Z}^2} = \frac{f + by}{\mathrm{Z}^2}$$

Eq. 13

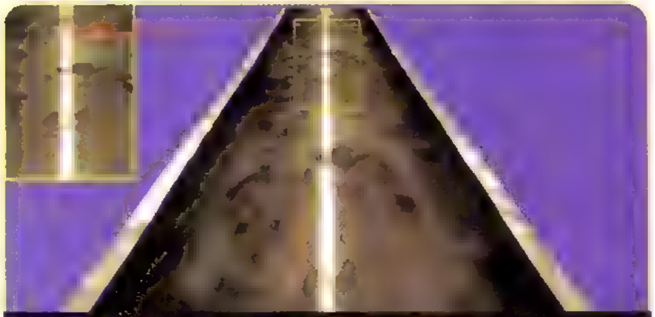


FIGURE 10. *Road rendered with per-pixel MIP-mapping and point sampling.*

Where *a*, *b*, *c*, *d*, *e*, and *f* are given by Equations 14 through 19.

$$a = \mathrm{dUOverZdX} * \mathrm{dOneOverZdY} - \mathrm{dOneOverZdX} * \mathrm{dUOverZdY}$$

Eq. 14

$$b = \mathrm{dVOverZdX} * \mathrm{dOneOverZdY} - \mathrm{dOneOverZdX} * \mathrm{dVOverZdY}$$

Eq. 15

$$c = \mathrm{dUOverZdX} * \mathrm{OneOverZ}_0 - \mathrm{dOneOverZdX} * \mathrm{UOverZ}_0$$

Eq. 16

$$d = \mathrm{dVOverZdX} * \mathrm{OneOverZ}_0 - \mathrm{dOneOverZdX} * \mathrm{VOverZ}_0$$

Eq. 17

$$e = \mathrm{dUOverZdY} * \mathrm{OneOverZ}_0 - \mathrm{dOneOverZdY} * \mathrm{UOverZ}_0$$

Eq. 18

$$f = \mathrm{dVOverZdY} * \mathrm{OneOverZ}_0 - \mathrm{dOneOverZdY} * \mathrm{VOverZ}_0$$

Eq. 19

An important point to note here is that the numerators of the partial derivatives u_x and v_x are functions of *y* only, and the numerators of the partial derivatives u_y and v_y are functions of *x* only. The values of *a*, *b*, *c*, *d*, *e*, and *f* are calculated once per polygon, along with the usual texture gradients, as shown in Listing 6. Finally, the formula for finding the maximum edge compression is given by Equation 20.

$$\mathrm{Compression} = \frac{1}{\mathrm{Z}^2} \max(x_i, y_i)$$

where

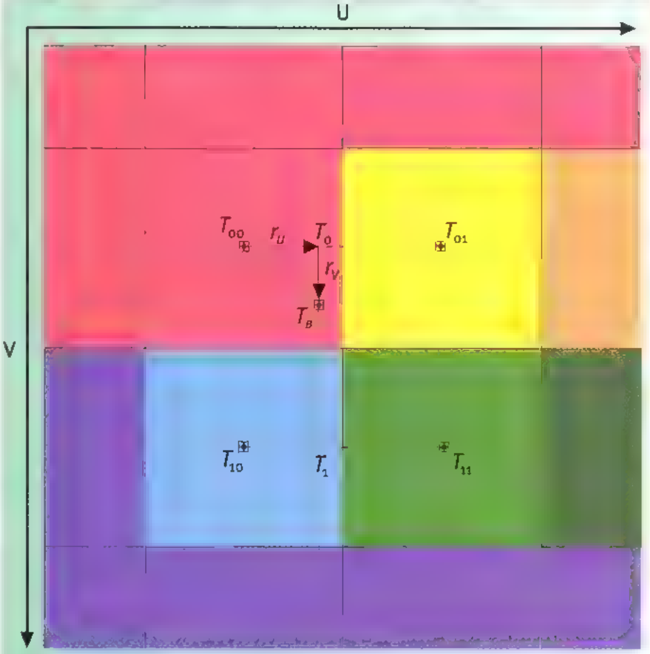
$$x_i = \left(\sqrt{(c = ay)^2 + (d + by)^2} \right)$$

$$y_i = \left(\sqrt{(e = ax)^2 (f \quad bx)^2} \right)$$

Eq. 20

At first glance, it would seem that we would need to carry out a square-root function at each pixel. However, if you look closely, you'll see that we only need to compute y_i once for the polygon's range of *x* values. Furthermore, we only need to compute x_i once per scan line. Listing 7 shows how we pre-compute the y_i values for a polygon's range of *x* values during the normal set-up for texture mapping, and also that we only calculate x_i once per scan line. We also don't have to worry

FIGURE 11. *Bilinear filtering calculation.*



$$T_0 = T_{00} + r_U(T_{10} - T_{00})$$
$$T_1 = T_{01} + r_U(T_{11} - T_{01})$$
$$T_B = T_0 + r_V(T_1 - T_0)$$

about the divide required for the denominator, because it's already required for standard texture lookup. So the overhead for the compression calculation within the texture-mapping inner loop is just two multiplies and a compare. Now that we know how to calculate the edge compression, let's apply per-pixel LOD selection to our texture-mapping routines using point sampling, bilinear filtering, and trilinear filtering.

POINT-SAMPLED PER-PIXEL MIP-MAPPING. Point sampling is the simplest form of per-pixel MIP-mapping, and as you can see in Listing 8, there isn't much difference between our normal texture-mapping loop and one that uses point sampling. Once we've found the amount of edge compression for the current pixel, we need to determine the correct LOD. The raw compression value ranges from a zero to one, but we need to scale it by the texture dimensions to get a meaningful height in our MIP-map pyramid. Once we have the height, we determine the correct LOD by stepping up one level in the pyramid for each power of two that the height is greater than one. We then use our fast LOD lookup table to get a pointer to our texture and access the correct texel as usual. Figure 10 shows the same object that we used to generate Figure 8, but this time we're applying point-sampled MIP-mapping. As you can see in the figure, the main problem with point-sampled MIP-mapping is that MIP-banding is clearly visible at the points where transitions between different LODs occur. This is because adjacent pixels can have different LODs, so a discontinuity appears as we switch between LODs.

BILINEARLY FILTERED PER-PIXEL MIP-MAPPING. Bilinear filtering attempts to further reduce any aliasing errors present in a scene by averaging the values of the four pixels that are closest to the real *u* and *v* texture values for each pixel. As you can see in Figure 11 and Listing 9, bilinear interpolation can be implemented using three linear interpolations. We calculate the correct LOD and retrieve the pointer to our texture in exactly the same way that we did with point sampling. However, we then retrieve four texture values and apply bilinear interpolation to each color component to generate the new pixel value. Figure 12 shows our road after MIP-mapping and bilinear filtering. Although Figure 12 is an improvement over Figure 10, you can still make out the MIP-banding. Nothing has been done to remove the discontinuities that occur when we switch between LODs.

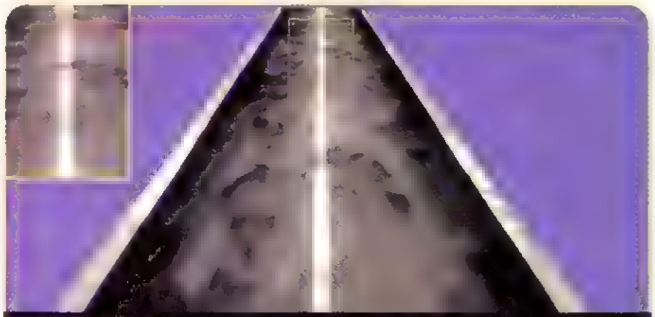


FIGURE 12. *Road rendered with per-pixel MIP-mapping and bilinear filtering.*

TRILINEARLY FILTERED PER-PIXEL MIP-MAPPING. The current state-of-the-art for 3D hardware-accelerated MIP-mapping is trilinear filtering. Trilinear filtering attempts to remove the problems associated with MIP-banding by smoothly blending between differing LODs. As you can see in Listing 10, we once again calculate the correct LOD in exactly the same way that we did it for point sampling, then retrieve pointers to the calculated LOD and the next lower LOD (the next level up in the pyramid). Trilinear interpolation is implemented using eight linear interpolations. We begin by carrying out bilinear interpolation separately for each of the selected LODs, then finish off by linearly interpolating between the two LODs. As you can seen in Figure 13, trilinear interpolation does result in a smooth transition between LODs (though the overall scene appears somewhat blurred). Unfortunately, this feature comes at a considerable cost: the straightforward implementation of trilinearly filtered MIP-mapping presented here requires eight texture accesses for each pixel and a considerable amount of computation. Although it's possible to cut down on the number of texture look-ups by saving texel values between loop

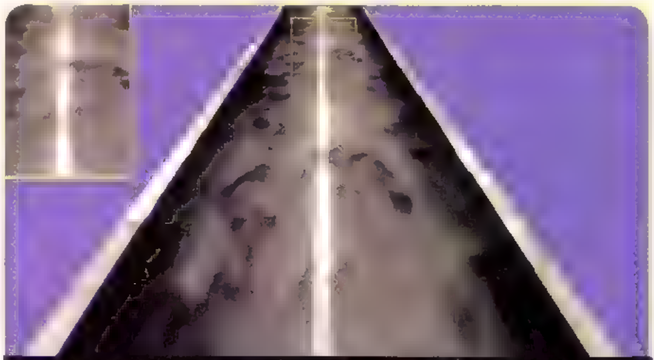


FIGURE 13. *Road rendered with per-pixel MIP-mapping and trilinear filtering.*

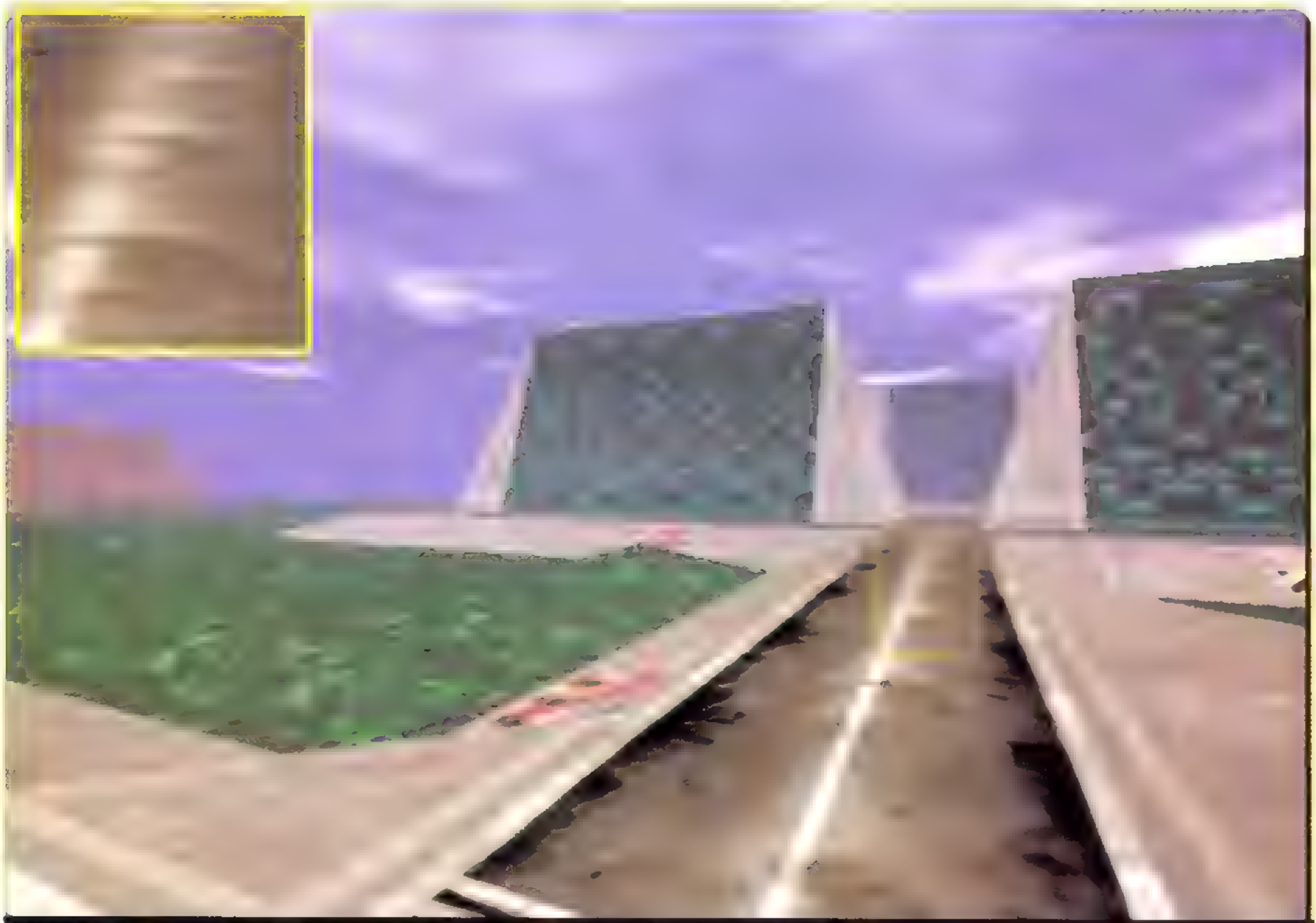


FIGURE 14. *Screen shot of ChaosVR rendered using trilinear filtering on a Voodoo2.*

iterations, the interpolations themselves need to be performed for each loop, so achieving acceptable frame rates with software-based trilinear filtering is very difficult.

We've covered a lot of ground for one article, and although the output of our renderer using trilinear MIP-mapping is significantly better than plain old texture mapping, it still isn't perfect. The biggest defect remaining in our filtering is that, as I mentioned earlier, we've ignored the fact that the texture compression is anisotropic. We're selecting LODs using the maximum compression along one edge, but what if there's a significant difference in the amount of compression between each edge? In this case, the LOD selected will be too low for the least compressed edge, and our scene will appear blurred. You can clearly see this effect in Figure 14, which is a screen shot from the ChaosVR demo that was rendered using a card based on 3Dfx's Voodoo2 chipset. This problem will occur with any 3D accelerator that uses methods similar to the ones that we've developed here for calculating the LOD – not just the Voodoo2 card that I'm using. Clearly, the next step to improve rendering accuracy will be to adopt some form of anisotropic filtering. I'm sure that it won't be long before this capability appears on high-end accelerators. ■

Bracewell, R. N., *The Fourier Transform and its applications*, McGraw-Hill Book Co., New York, 1986.

Williams, L., "Pyramidal Parametrics," Computer Graphics, vol. 17, no. 3, (Proc. SIGGRAPH 1983).

Heckbert, P., "Texture Mapping Polygons in Perspective," NYIT Tech. Memo No. 13, 1983

Thanks go out to Chris Hecker who kindly allowed me to plug my MIP-mapping into his texture mapping routines, saving me a lot of time. Check out Chris's home page, , for more information on texture mapping and his old columns from *Game Developer*.

I'd also like to thank Paul Heckbert for taking the trouble to send me one of the first publications to ever discuss MIP-mapping. You can also find a lot of information about texture mapping and myriad other graphics techniques on Paul's home page .

Finally, I'd like to thank Peter Laufenberg for allowing me to use a screen shot from Virtually Unlimited's ChaosVR demo. You can find out more about the demo at .

A Simple, Fast, and Effective Polygon Reduction Algorithm

by Stan Melax

If you're a game developer, there's a good chance that 3D polygonal models are part of your daily life and that you're familiar with concepts such as polygons per second, low-polygon modeling, and levels of detail. You probably also know that the objective of a polygon reduction algorithm is to take a high-detail model with many polygons and generate a version using fewer polygons that looks reasonably similar to the original. In addition to talking about what polygon reduction is and why it is useful, this article explains one method for achieving it. Before going any further, I suggest you download my application, BUNNYLOD.EXE, which demonstrates the technique that I'll explain. You can find it on the *Game Developer* web site.

Motivation

Before diving into a sexy 3D algorithm, you may be asking yourself if you really care. After all, there are commercial plug-ins and tools that reduce polygons for you. Nonetheless, there may be reasons why you want to implement your own reduction algorithm.

- The results of your polygon-reduction tool may not meet your specific needs, and you would like to build your own.
- Your current polygon-reduction tool may not produce the morph information that you require for smooth transitions between different levels of detail.
- You want to automate your production pipeline so that the artist has to create only one reasonably detailed model, and the game engine does the rest.
- You're creating a VRML browser, and you want to provide a menu option for reducing those huge VRML files placed on the Web by supercomputer users who didn't realize the frame rate would be slower on a home PC.
- Special effects in your game modify the geometry of objects, bumping up your polygon count and requiring a method by which your engine can quickly reduce polygon counts at run time.

Stan Melax is researching interactive 3D techniques and algorithms for his Ph.D. in computer science at the University of Alberta. He is also the Director of Technology at Bioware, where he had worked on SHATTERED STEEL *and is now implementing cool stuff for their next 3D titles. He can be contacted via e-mail at melax@cs.ualberta.ca.*

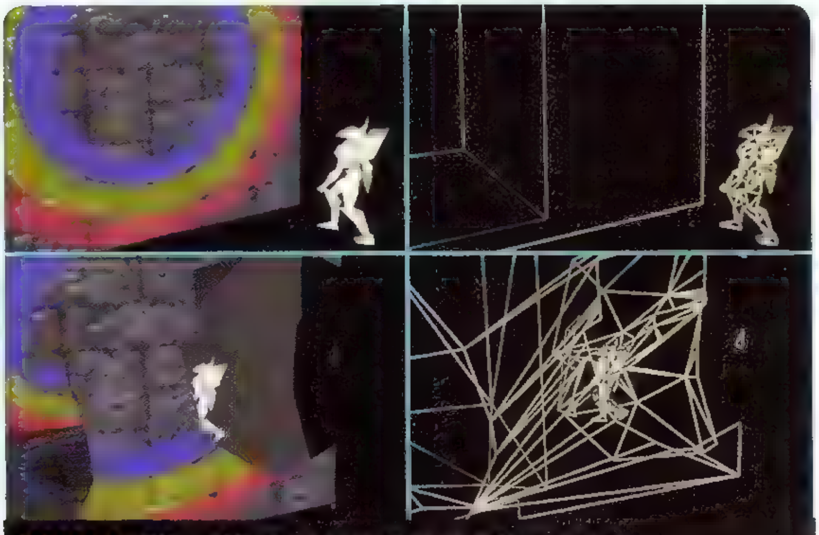


FIGURE 1. *Effect of Boolean operations on polygon count.*

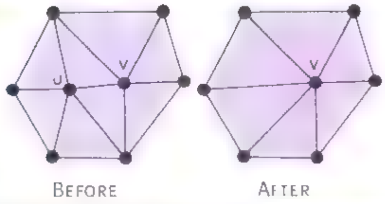


FIGURE 2. *Edge collapse.*

Still not convinced? Figure 1 shows a concrete example of an instance in which a game engine requires polygon reduction capabilities.

At Bioware, I implemented real-time Boolean operations and used them in a game prototype that we developed to impress our publisher. Players could shoot and blast arbitrary chunks out of a solid object wherever they decided to point the gun. Modifying the game environment where the bullets impact produces much more stunning results than the typical "place pipe bomb here" technique, in which the game world only changes in a predetermined manner. Unfortunately, repeated use of Boolean operations performed on polygonal objects generates lots of additional triangles, as you can see in Figure 1. Many of these additional faces are small or splinter triangles that don't contribute to the visual quality of the game — they just slow it down. The situation demanded run-time polygon reduction, so I began my quest to find an algorithm that would do this efficiently.

Collapsing Edges

Rather than attacking this problem all by myself, I studied polygon reduction with some other people at the University of Alberta Graphics Lab. (It helps to work with a team in order to figure out how the different algorithms work and which technology is appropriate for which task.) A lot of research has gone into this subject recently, and most of the better techniques are variations of the progressive meshes algorithm by H. Hoppe (see "For Further Info"). These techniques reduce a model's complexity by repeated use of the simple edge collapse operation, shown in Figure 2.

In this operation, two vertices *u* and *v* (the edge *uv*) are selected and one of them (*u*) is "moved" or "collapsed" onto the other (in this case, *v*). The following steps implement this operation:

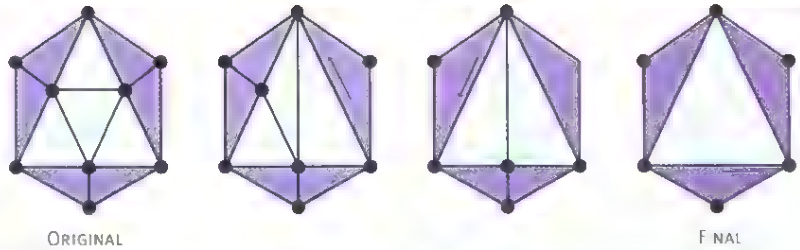
1. Remove any triangles that have both *u* and *v* as vertices (that is, remove triangles on the edge *uv*)
2. Update the remaining triangles that use *u* as a vertex to use *v* instead.
3. Remove vertex *u*.

This process is repeated until the desired polygon count is reached. At each step, one vertex, two faces, and three edges are usually removed. Figure 3 shows a simple example.

Selecting the Next Edge to Collapse

The trick to producing good low-polygon models is to select the edge that, when collapsed, will cause the smallest visual change to the model. Researchers have proposed various methods of determining the "minimal cost" edge to collapse at each step. Unfortunately, the best methods are very elaborate (as in, difficult to implement) and take too long to compute. Motivated to find a way to reduce polygons during run time in a game, I performed many experiments and eventually developed a simple and blazingly fast approach for this selection process that generates reasonably good low-polygon models.

FIGURE 3. *Polygon reduction via a sequence of edge collapses.*

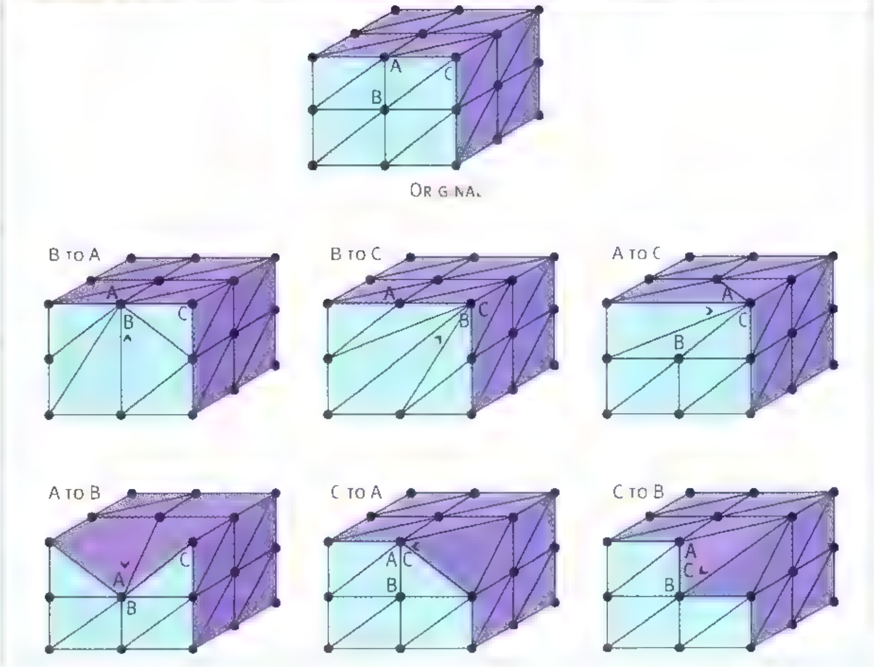


EQUATION 1. *The edge cost formula.*

$$\text{cost}(u,v) = \|u - v\| \times \max_{f \in Tu}\left\{\min_{n \in Tuv}\left\{(1 - f.normal \bullet n.normal) \div 2\right\}\right\}$$

where *Tu* is the set of triangles that contain *u* and *Tuv* is the set of triangles that contain both *u* and *v*.

FIGURE 4. *Good and bad edge collapses.*



Obviously, it makes sense to get rid of small details first. Note also that fewer polygons are needed to represent nearly coplanar surfaces while areas of high curvature need more polygons. Based on these heuristics, we define the cost of collapsing an edge as the length of the edge multiplied by a curvature term. The curvature term for collapsing an edge *uv* is determined by comparing dot products of face normals in order to find the triangle adjacent to *u* that faces furthest away from the other triangles that are along *uv*. Equation 1 shows the edge cost formula in more formal notation. The specific details can also be found in the source code (which you can dowlnoad from *Game Developer*'s web site).

You can see that this algorithm balances curvature and size when determining which edge to collapse. Note that the cost of collapsing vertex *u* to *v* may be different than the cost of collapsing *v* to *u*. Furthermore, the formula is effective for collapsing edges along a ridge. Although the ridge may be a sharp angle, it won't matter if it's running orthogonal to the edge. Figure 4 illustrates this concept. Clearly, vertex B, sitting in the middle of a flat region, can be collapsed to A or C. Corner vertex C should be left alone. It would be bad to move vertex A, sitting along the top ridge, onto interior vertex B. However, A could be moved (along the ridge) onto C without affecting the overall shape of the model.

If you're implementing your own reduction algorithm, you may wish to experiment with this equation in order to meet your needs. For example, in the case of an animating mesh, you might want to develop a formula that will look at more than just one keyframe when computing the cost of a potential edge collapse. If quality is more important to you than the reduction algorithm's execution time, then you should consider using Hoppe's energy function. We've added our own extensions to deal with texture coordinates, vertex normals, border edges, and surface discontinuities such as texture seams.

Results

The effectiveness of a polygon reduction algorithm is best demonstrated by showing a model before and after it has been simplified. Most research papers demonstrate their results using highly tessellated models in the neighborhood of 100,000 polygons, reducing them to 10,000 polygons. For 3D games, a more appropriate (and challenging) test of an algorithm is how it demonstrates its prowess by generating models that use only a few hundred polygons.

For instance, Figure 5 shows a bunny model taken from a VRML file created by Viewpoint Datalabs. The initial version (left) of the model contains 453 vertices and 902 polygons. Reductions to 200 (center) and 100 (right) vertices are shown. Hopefully, you'll agree that the models look reasonably good given the number of polygons used in each image. Figure 6 shows the consequences of not selecting the right edge to collapse at each step. In this case, edges were chosen randomly.

After completing animal testing, we began human clinical trials for the algorithm. Figure 7 shows three versions — at 4,858; 1,000; and 200 vertices — of a

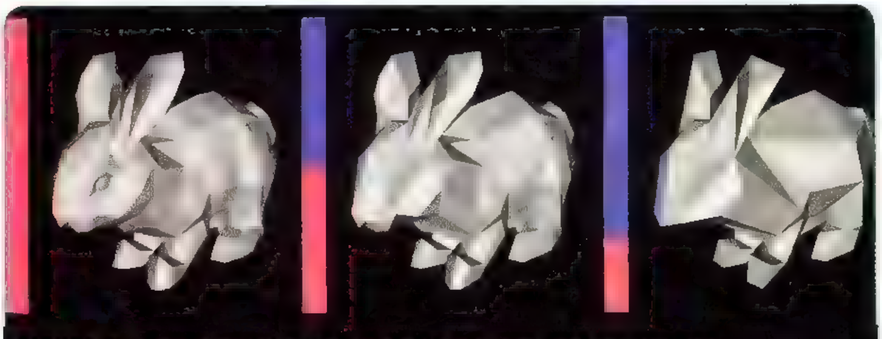


FIGURE 5. *Bunny model at (left to right) 453, 200, and 100 vertices.*

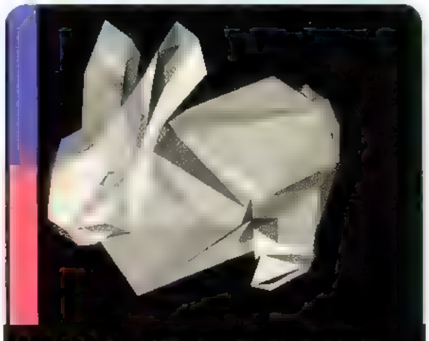


FIGURE 6. *Random edge selection (200 vertex version).*

female human model made by Bioware. (From Euler's formula, we know that the polygon counts are roughly double these numbers.) Once again, these images are shown with flat shading so you can see the difference in the meshes. When smooth shading and textures are applied, the differences are less apparent

Practical Application

Our initial goal was modest: we wanted to find a way to get rid of a few excess polygons caused by too many Boolean operation effects. However, after developing the reduction algorithm and noticing better-than-expected results on actual models, we decided that the technique was good enough to generate the level of detail (LOD) models for the game engine. An improved version of this basic algorithm has since been incorporated into Bioware's 3D graphics engine, Omen. Now, for many game objects, our artists only have to create one detailed model. A preprocessing step does the polygon reduction. Then, when the frame rate falls below a predefined threshold or an object is to be rendered in the distance, a lower polygon version is used instead. Being able to make these choices at run time increases the scalability of a game. The game adapts itself to the horsepower of the system on which it's running.

Implementation Details

This algorithm only works with triangles. Nothing is lost by this limitation; polygons with more sides are easily triangulated if necessary. In fact, many applications use triangles exclusively.

Most data structures for storing polygonal objects use a list of vertices and a list of triangles that contain indices into the vertex list. For example,

```
Vector vertices[];
class Triangle {
   int v[3];   // indices into vertex list
} triangles[];
```

The `Indexed Face Set` node data type used in VRML is another example of this type of data structure. When two

LISTING 1. *The enhanced data structure.*

```
class Triangle {
   public:
         Vertex *   vertex[3]; // the 3 points that make this tri
         Vector     normal;    // orthogonal unit vector
                    Triangle(Vertex *v0,Vertex *v1,Vertex *v2);
                    ~Triangle();
         void       ComputeNormal();
         void       ReplaceVertex(Vertex *vold,Vertex *vnew);
         int        HasVertex(Vertex *v);
};
class Vertex {
   public:
         Vector     position; // location of this point
         int        id;       // place of vertex in original list
         List<Vertex *>    neighbor; // adjacent vertices
         List<Triangle *>  face;     // adjacent triangles
         float      cost;     // cached cost of collapsing edge
         Vertex *   collapse; // candidate vertex for collapse
                    Vertex(Vector v,int _id);
                    ~Vertex();
         void       RemoveIfNonNeighbor(Vertex *n);
};
List<Vertex *>      vertices;
List<Triangle *>    triangles;
```

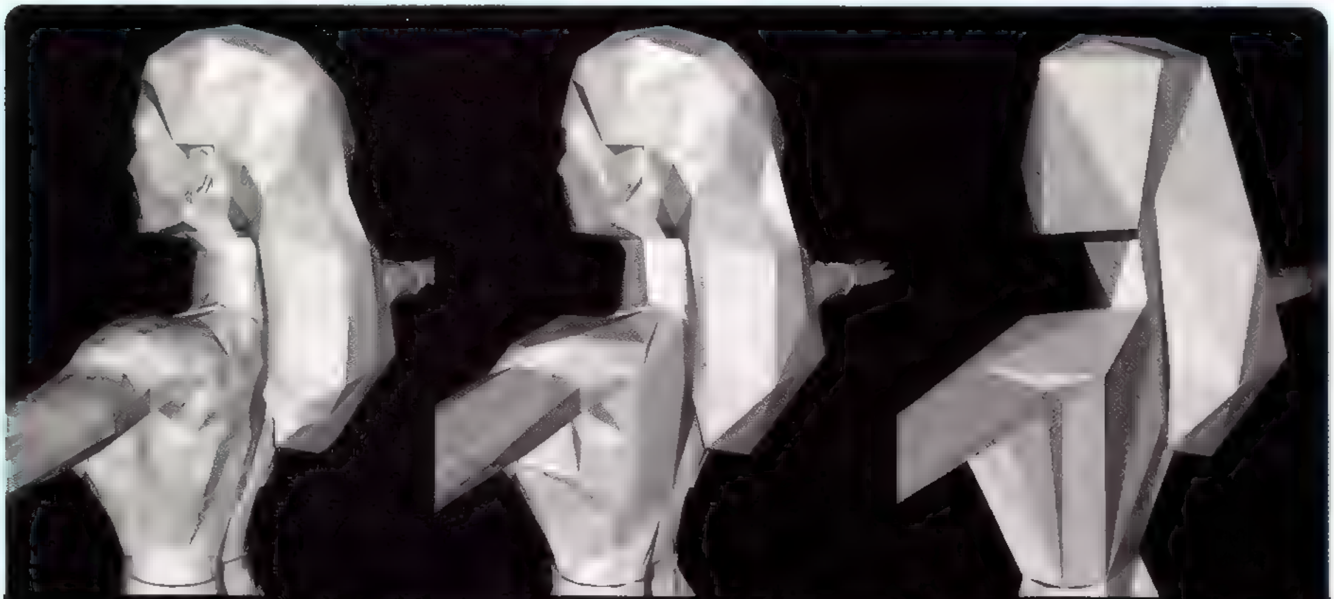


FIGURE 7. *Female human model showing 100 percent of the original polygons (left), 20 percent of the original polygons (center), and 4 percent of the original polygons (right).*

triangles on an object meet at the same vertex, they'll have the same index (so they share the same entry in the vertex list).

We've enhanced this data structure as required by our polygon reduction algorithm. One major improvement is that we now have access to more information than just which vertices each triangle uses — we also know which triangles each vertex bounds. Furthermore, for each vertex, we have direct access to its neighboring vertices (which gives us the edges). Listing 1 shows the enhanced data structure.

Member functions such as `ReplaceVertex()` have been added to perform edge collapses during polygon reduction. Consistency of this data must be maintained as vertices and triangles are added, deleted, or replaced. The constructors, destructors, and member functions contain code to keep things in order. We cache face normals because they are frequently used by the edge selection formula. In order to save us the effort of recalculating these costs, the best edge and its cost is cached for each vertex. The implementation of the member functions is fairly straightforward, so I haven't included it in this article. If you're interested, simply examine this algorithm's source code on the *Game Developer* web site. Listing 2 contains the code for determining edge costs and doing the edge collapse operation.

Performing polygon reduction is easy given these functions. Simply initialize the vertex and triangle lists with the object's geometry, and then do something like this:

```
while(vertices.num > desired) {
    Vertex *mn = MinimumCostEdge();
    Collapse(mn,mn->collapse);
}
```

The demo, BUNNYLOD.EXE, doesn't use this simple loop. Instead it creates an additional data structure for the animation.

Making Better Use of the Data

Rather than throwing away information about triangles and vertices that have been removed, this information can be preserved so that a

LISTING 2. *Determining the edge costs and performing the edge collapse operation.*

```
float ComputeEdgeCollapseCost(Vertex *u,Vertex *v) {
    // if we collapse edge uv by moving u to v then how
    // much different will the model change, i.e. the "error".
    float edgelength = magnitude(v->position - u->position);
    float curvature=0;

    // find the "sides" triangles that are on the edge uv
    List<Triangle *> sides;
    for(i=0;i<u->face.num;i++) {
        if(u->face[i]->HasVertex(v)){
            sides.Add(u->face[i]);
        }
    }
    // use the triangle facing most away from the sides
    // to determine our curvature term
    for(i=0;i<u->face.num;i++) {
        float mincurv=1;
        for(int j=0;j < sides.num;j++) {
            // use dot product of face normals.
            float dotprod =
               u->face[i]->normal ^ sides[j]->normal;
            mincurv = min(mincurv,(1-dotprod)/2.0f);
        }
        curvature = max(curvature,mincurv);
    }
    return edgelength * curvature;
}

void ComputeEdgeCostAtVertex(Vertex *v) {
    if(v->neighbor.num==0) {
        v->collapse=NULL;
        v->cost=-0.01f;
        return;
    }
    v->cost = 1000000;
    v->collapse=NULL;
    // search all neighboring edges for "least cost" edge
    for(int i=0;i < v->neighbor.num;i++) {
            float c;
            c = ComputeEdgeCollapseCost(v,v->neighbor[i]);
            if(c < v->cost) {
                v->collapse=v-neighbor[i];
                v->cost=c;
            }
    }
}

void Collapse(Vertex *u,Vertex *v){
    // Collapse the edge uv by moving vertex u onto v
    if(!v) {
        // u is a vertex all by itself so just delete it
        delete u;
        return;
    }
    int i;
    List<Vertex *>tmp;
    // make tmp a list of all the neighbors of u
    for(i=0;i<u->neighbor.num;i++) {
        tmp.Add(u->neighbor[i]);
    }
    // delete triangles on edge uv:
    for(i=u->face.num-1;i>=0;i--) {
        if(u->face[i]->HasVertex(v)) {
            delete(u->face[i]);
        }
    }
    // update remaining triangles to have v instead of u
    for(i=u->face.num-1;i>=0;i--) {
        u->face[i]->ReplaceVertex(u,v);
    }
    delete u;
    // recompute the edge collapse costs in neighborhood
    for(i=0;i<tmp.num;i++) {
        ComputeEdgeCostAtVertex(tmp[i]);
    }
}
```

model at any specified number of vertices can be retrieved on demand without having to recompute the polygon reductions. This feature is easily implemented by storing the vertex to which each vertex is collapsed and sorting the vertices by the order in which they were collapsed.

The BUNNYLOD.EXE demo uses this method. Initially, the bunny is reduced from 450 to 0 vertices in approximately one second. Then, as the slider on the left animates the bunny, the model is rendered in increasing detail using the specified number of polygons. Another way to think of this animation is as a sequence of models for every number of vertices between 0 and the number in original model.

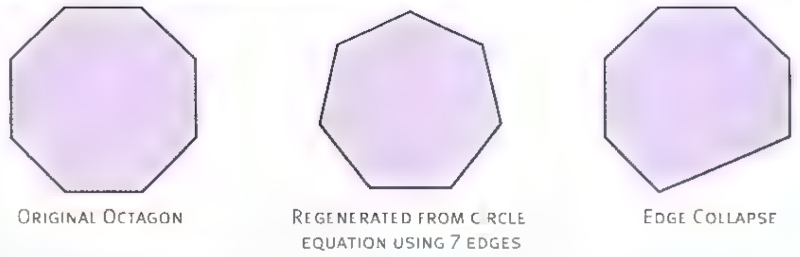
The edge collapse sequence could also be used for progressive transmission. Just as interlaced .GIF and .JPG pictures come over the Web in increasing detail, the vertices of an object can be broadcast in the reverse order from which they were collapsed. The receiving computer can display the model while it is reconstructed from the incoming data stream. This is a nice idea, but it's probably not relevant for game developers just yet.

An important component in many games is the LOD of models. A handful of models can be selected from the sequence generated by our algorithm to represent the object at various LODs. One problem with swapping models is that players often notice when this occurs (the phenomenon known as "popping"). A solution to the popping effect is to morph smoothly between the models. In order to morph between two models, the vertices of one model must be mapped onto the other. Fortunately, this information can be extracted from the edge collapse sequence. The BUNNYLOD.EXE demo also shows an example of morphing.

Alternatives to Edge Collapse Techniques

Polygon reduction algorithms aren't the only way to create a model with fewer faces. Artists will always be able to do a better job of representing a model using fewer polygons than any reduction algorithm. One reason is that algorithms have little or no higher-level understanding of the model. An artist, on the other hand, knows the object that he or she is creating (be it a rabbit, a chair, and so on) and can make careful aesthetic decisions as he or she manually reduces the face count. The human visual system is biased towards certain details, such as the eyes and mouth, and pays less attention to other details such as the collarbone or kneecaps. On the other hand, our simple algorithm merely compares a few dot products and edge lengths, and obviously doesn't have the intelligence to place automatically varying amounts of importance on different pieces to optimize for human perception. The advantage to using a polygon reduction algorithm is that it automates the process.

FIGURE 8. *Comparison of techniques.*



Another technique for doing LODs in a game is to represent an object's geometry using parametric surface patches, which are tessellated on the fly to the desired detail. Shiny's MESSIAH engine uses a similar approach. Certainly, these surface-based methods are preferable (and probably optimal too). Figure 8 illustrates the advantage using a 2D analogy. An octagon reduced by one edge is regenerated as a regular heptagon by the parametric approach. Collapsing an edge on the octagon produces non-regular results.

Unfortunately, using curved parametric surfaces isn't always appropriate. Some of the challenges include getting the object into this sort of representation and being able to generate polygons at render time so that adjacent surfaces fit together properly (without gaps or T-intersections). Furthermore, jagged objects aren't good candidates for use with curved surface patches because the number of surfaces would be no less than the number of polygons required. Polygon-based reduction methods are more generally useful, and work with typical models used these days.

While I hope that this information and the accompanying demonstration application that I've provided are useful, this article has not touched on issues such as dealing with texture coordinates, vertex normals, border edges, nonmanifold topology, texture seams, and so on. These subjects have been left as an exercise for the reader. Furthermore, many other variations and enhancements to this algorithm are worth exploring. One exciting topic is adaptive simplification, in which different parts of the same mesh are rendered at different levels of detail according to run-time parameters. This is especially useful for open terrain environments so that more detail can be used near the current viewpoint. ■

[illegible]

Polygon reduction has been a hot research topic lately, and most of the literature about it can be found in proceedings from academic computer graphics conferences. Some more places you can look:

- Cohen, J., M. Olano, and D. Manocha. "Appearance-Preserving Simplification", SIGGRAPH '98.
- Hoppe, H. "Progressive Meshes," SIGGRAPH '96, pp. 99-108.
- Luebke, D. and C. Erikson. "View-Dependent Simplification of Arbitrary Polygonal Environments", SIGGRAPH '97, pp. 199-207.
- I have a demo on my university web site at http://www.cs.ualberta.ca/~melax/polychop
- H. Hoppe, the Guru of polygon reduction, maintains a web site at http://research.microsoft.com/~hoppe

Experience the be-all, end-all online resource for the game development community...

gamasutra.com

The Art and Science of Making Games

sponsored by

COMPUTER GAME DEVELOPERS CONFERENCE™

ON THE FRONT LINE OF GAME INNOVATION
Game DEVELOPER

Animating Facial Expressions

by Jake Rodgers

The challenge of capturing the subtleties of facial movement is what brought me back to motion capture after a brief but scarring first experience and, consequently, what compelled me to write about my trials. Now that I've taken my lumps, I'm in a good position to help you understand your options and teach you a bit about the process itself.

First, let me say that you'd have to be crazy to try to animate a face to match a voice. Entire regions of the human brain are dedicated to recognizing facial patterns and body language. Hundreds of muscles in a human face move, contract, and slide in and out of one another, mocking our attempts to duplicate their deceptively simple duties. One simple mistake can transform your life-like character into an evil pile of face parts and prompt reactions such as, "Ooh, that's disturbing," from passersby.

It didn't take much for me to realize why I'd never seen a convincing realistic 3D head model talking, especially one composed of only a few hundred polygons. After being briefly exposed to the technology and noting the queer gestures and difficult timing problems of facial animation, I felt that the technology simply wasn't ready.

But if you have to make it work, and we had to make it work, what can you do? We had a very large script with many medium and close-up shots, and I had to come up with a way to animate pages of script spoken and acted out by low-polygon characters.

Image courtesy of Pyros Pictures.

Approaches

You can approach facial animation in any one of several ways; you'll need to decide early on which way you'd like to go. Probably the most flexible and powerful of these is the approach used by Pacific Data Images in the upcoming DreamWorks film *Antz*. This solution involves researching all the muscles of the face and using a huge team of artists and programmers for a couple of years to develop an animation system based on the way human facial muscles move and slide on the skull. You probably don't want to approach the problem in this way, however. A more reasonable approach is

Jake Rodgers has earned the prestigious title of "One of many" at Digital Anvil, and his dream is to art direct the Second Coming. His first game experience was WING COMMANDER II, VENGEANCE OF THE KILRATHI, *and he has a feeling his career is "about to really take off." He can be reached at jake@digitalanvil.com*

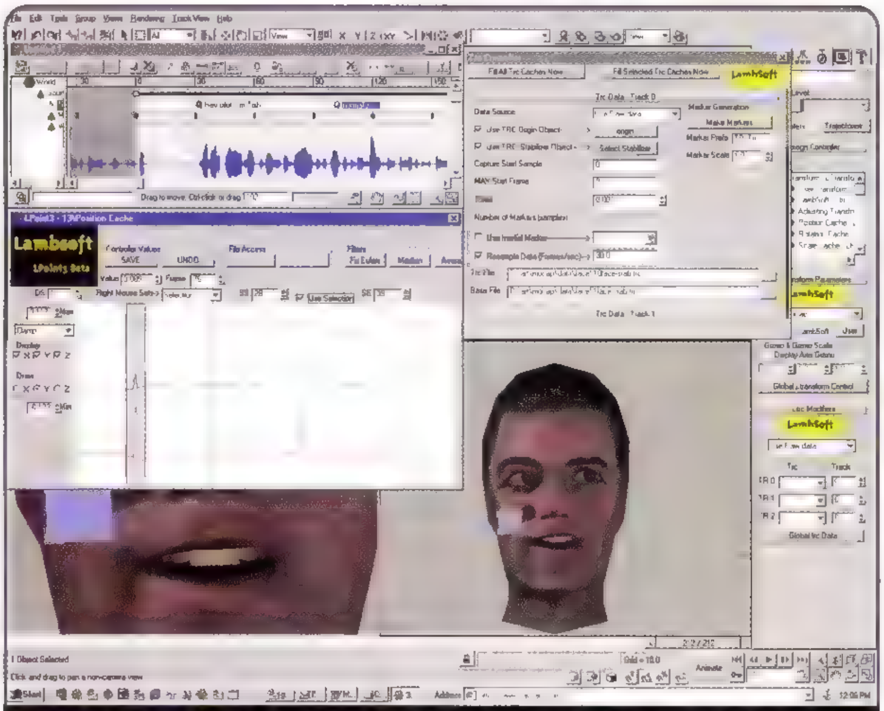


FIGURE 1. *Lambsoft's ProMotion lets the user actually draw over the function curves for markers. This made cleaning up spikes in the data fairly easy. ProMotion also lets the user apply global filters such as median and average, as well as a Fix Eulers filter that takes away unwanted flips in rotation data for body motion capture. ProMotion is very good at taking motion capture data and putting it on characters whose proportions don't match the actor. The screenshot is from a beta release of ProMotion.*

to use either some form of phoneme target blending, 2D or 3D motion capture, or a combination of these. I will explain each of these methods briefly just in case you aren't familiar with them.

Phoneme Blending

Webster defines phoneme as, "Any of the abstract units of the phonetic system of a language that correspond to a set of similar speech sounds (as the velar \k\ of cool and the palatal \k\ of keel) which are perceived to be a single distinctive sound in the language."

There are many different ways to blend phonemes in an animated character, but the vast majority of software tools share the same process. The process involves creating a template of phonemes, or gestures, then assigning each of them to specific MIDI channels, which are in turn controlled by slider controls. The position of each slider determines the percentage of that phoneme's contribution to the overall gesture. A channel updates only the vertices of the polygon in the face used with a particular phoneme's gesture, allowing you to mix a frown with a yawn, or a wink with a smile, and so on. Plug-ins such as Lambsoft's Smirk and Platinum Pictures' MorphMagic work in this way. Kaydara's FiLMBOX has a feature called Voice Reality that can recognize phonemes from a voice track and lip-sync the animation for you.

Phoneme blending is a way to make sure your gestures look correct (assuming that you have a good template of phonemes) because it allows you to interactively see the results. You can then say, for example, "Hmm, it needs more frown" or "More 'ooooohh' please." Unfortunately, this process leaves the burden of timing on your shoulders. Animators may be reading this and saying to themselves, "No problem. I'm an animator, after all." However, remember that my project involved a large script that would have driven even the purest animator nuts trying to keyframe every sequence using these kinds of tools.

Phoneme blending is a good solution if you want the ability to change the timing of the voice after you've animated the face, if you have quick animations, if you're working under a tight budget, or if you really enjoy phoneme blending.

2D Facial Capture

Two-dimensional facial capture is a widely used, efficient way to apply motion to a character's face. The hardware typically used for 2D facial capture is Adaptive Optics Associates' FaceTrax, Motion Analysis's FaceTracker, and Vierte Art's Xist. These products use head-mounted devices that track markers applied to an actor's face, usually using one camera. Two-dimensional facial capture is a good method for performance animation and near-real-time animation.

Another method uses a regular video camera and a tool such as Techimage's Artiface. This tool uses a sophisticated edge-detection program to track the movement of the lips (and only the lips) of a filmed actor. The user then imports a 3D object, applies some simple muscle structure to the object to control the way the mesh is deformed, and then applies the captured animation.

If your character is going to be looking directly at the camera, and not generally seen at different angles, 2D capture might be suitable for your project. However, this method is limited in that it really can't capture subtleties on the face, and it can't accurately reproduce what's happening in the z axis (which is very important for smiles).

The other problem with 2D facial capture is that the head-mounted cameras can hamper the actor's natural motion, and you want him or her to be comfortable (especially during lengthy capture sessions). I'm sure there are lighter units coming out on the market, but considering the aforementioned cons and the particulars of our project, 2D facial capture was not a good solution for us.

3D Facial Capture

Three-dimensional facial capture gives you the most movement information, including head movement and almost any facial gesture your talent can muster. It also gives you the ability to match up, with relative ease, any actor's face with a computer generated character (I'll explain this process shortly). Although this approach also poses the most potential problems, at least you're capturing a great deal of data. With the right mix of accurate captures and good clean-up software, you'll be able to use this data once you're back in your office.

One of the known problems of 3D facial capture is that it's difficult to manipulate the geometry of a face once you've applied the motion to it. One process I'm currently trying to develop is a hybrid of 3D capture and phoneme blending. During the capture session, the face actor is asked to go through the typical phonemes — frowns, smiles, and so on — from which we would construct a base phoneme library. Each gesture would have a keyframe for each marker, which could be layered upon the capture data. Just as with phoneme blending, we would assign each marker to a MIDI channel. Now, if we need the character's mouth to shut more, we simply adjust the appropriate sliders, which in turn moves the markers a certain percentage of the way towards the keys we had previously set.

Tools

Several tools are available to game developers trying to achieve good facial animation results. And even though it might not be a good idea for someone to stick their neck out and rant about tools, I did a modest amount of research on every major solution I could find.

Right now, I would argue that the combination of Kinetix's 3D Studio MAX, Character Studio, and Lambsoft's ProMotion (Figure 1) makes the most sense on several levels. The most obvious advantage is cost. Most comparable packages (such as Alias|Wavefront's PowerAnimator or Maya in its impressive, but young form) would have created a budget problem for us, especially considering that our desired result was a real-time animated object with a low-polygon count. To us, the tool(s) in question had bend to the specific needs of our game engine, yet still have sufficient power and a relatively short learning curve. MAX had the benefit of a few years of real customer feedback and some very useable plug-ins, which made it the winner for us.

Another noteworthy tool is Sven Technology's SurfaceSuite Pro. It's perfect for creating head textures, using multiple projections on one object (top, sides, front, back), and masking projection layers so that you can blend several photographs (for example, a front view, a side view, and so on) seamlessly onto a head (Figure 2). SurfaceSuite then uses its own renderer to create a single, seamless cylindrical texture for real-time export, asking only that you define the resolution.

Process

The preproduction stage is critical to a project, particularly if you're going to employ a nascent technology such as facial motion capture. Figure 3 illustrates our development process workflow for everything relating to facial animation, from concept to game engine. Note that everything sprouted from the script and storyboards. A little inspirational art before that stage is helpful, but in our company, no true production is allowed until the producer signs off on concepts, scripts, and storyboards. (Of course, in practice, we've never strictly adhered to this rule. But you must agree it is a wonderful model to which to aspire.)

[illegible]

Once you have a good script and your characters have been fleshed out visually, the next step is to get voice talent. For a number of reasons, it's likely that the voice actor and the face model won't be the same person. In this case, it's best to get the audio recording out of the way first, and let the facial actor synchronize his or her gestures to the prerecorded voices during the motion capture session – in other words, lip sync. If possible, find someone who's done facial motion capture before, and knows exactly what's expected. Directing the talent to over-exaggerate all motions, including eyebrow activity, worked well for us. Overacted movements are much easier to remove, modify, or subdue than trying to exaggerate a subtle performance.

Because one group of people will produce the audio and another team will work on the motion capture data,

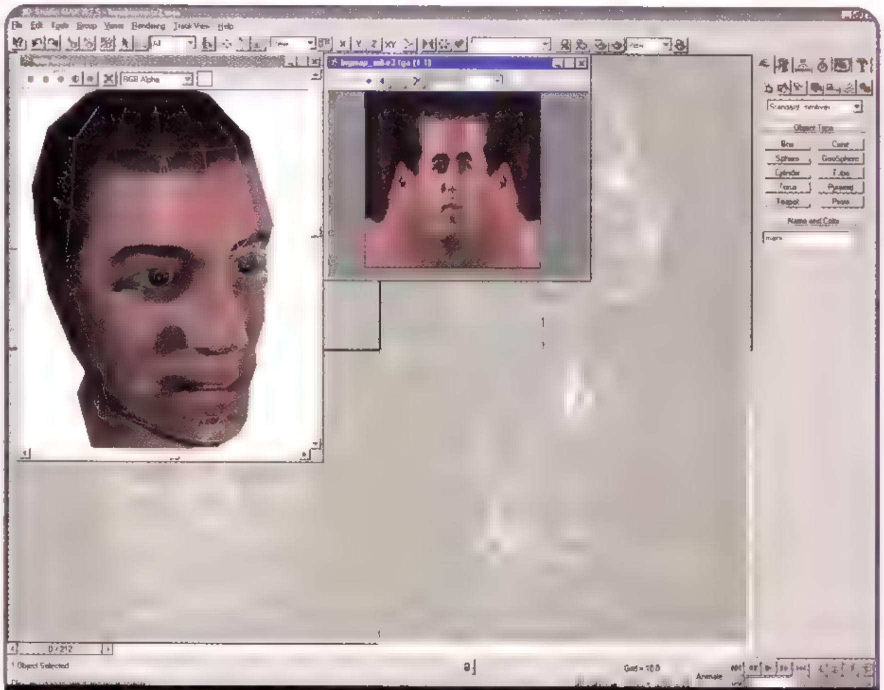
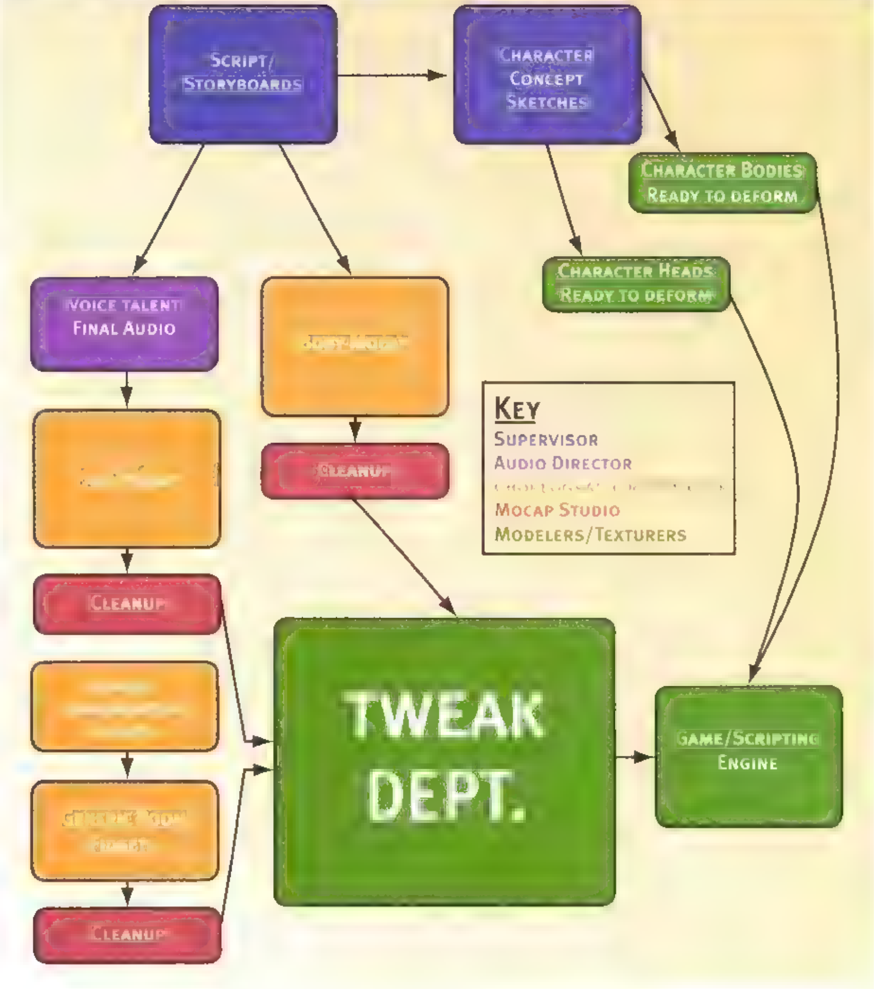


FIGURE 2. *This head was created to get a low polygon count (about 700) and to limit anomalies when animating. Kelcey Privett created the texture using three photos and SurfaceSuite. There are few things more frustrating than a face that almost works.*

FIGURE 3. *Sample workflow.*



make sure you chop up the script into small pieces. The captures should be short enough that the talent can easily perform sections in one or two takes, but not so short that you have millions of individual captures (note that some motion capture studios charge by the capture). One or two sentences per capture seems to be a good rule of thumb.

A good optical motion capture system — such as Motion Analysis's system — lets you trigger a capture from a .WAV file. For example, after we break up the script into separate audio files, we add three audible beeps one second apart before the voice recording. The third beep is at 500Hz, which activates the motion capture. Another 500Hz beep is played after the actor stops talking, automatically stopping the capture session. These beeps help the actor practice ahead of time, and once you get the data back, the audio is already matched up to your capture. This method saves huge amounts of time later on trying to synchronize the audio with the motion capture data.

I recommend that you send your face actor the script and audio tracks as early as possible. It really helps to have some kind of description of the characters you want them to portray. One of our actors wanted to know a lot about the characters beforehand. Initially, I felt that this person was just being difficult. I thought that giving the actor the voice track would suffice. Now that I've had a chance to look at the motion capture data, I understand a little more about the subtleties that can make a gesture either look believable or totally disturbing. In fact, now I'd even recommend videotaping the voice actor's recording sessions so the face actor can review them.

Make Your Face

Adding detail where it's most needed is essential for animating. Before you create the face mesh, consider how your character will animate. This advice may sound completely obvious, but it's important to add nonetheless. The fundamental thing to remember is to check all the phonemes, emotions, and proper blinking against your model (Figure 4). One good way to do this is have a face calibration motion capture file, which consists of your face actor going through a series of phonemes and extreme facial positions, so you can see where your face needs editing. Ideally, in the near future we'll be using patches to animate faces smoothly and to do automatic levels of detail. Don't count on your first low-detail head to work with the captures you get; the initial calibration is more of a test-and-see-what works kind of process.

Put the Motion on Your Skeleton

It would be nice if there were a very simple way to deform your low-resolution mesh, but there isn't. Many tools out there help the process, but complications always arise. We faced a number of challenges when we tried to apply our motion capture data to our geometry.

First, it became obvious to us was that no matter how hard the actor tried, the starting position for each capture was not constant. Luckily, I was working closely with Jeff Thingvold, chief scientist at Lambsoft, and he came up with a way to create a basis file that allowed us to base a series of animations on a given start frame. As long as the markers start in pretty much the same spot, they will work.

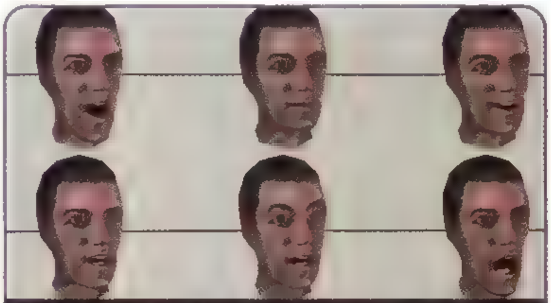


FIGURE 4. *Checking the phonemes to make sure the face will hold up.*

The flip side to having complete 3D data from a capture is that you need a way to separate out the whole head's motion from the just facial expressions. ProMotion can compare the positions of the markers and subtract the motion that all markers have in common so that only the relative face motion remains. Later, you can take this whole head motion and apply it to your neck bone separately.

We had a problem with head shapes. The game characters had differently shaped heads, and none of them resembled the shape of our face actor's head. To remedy this discrepancy, we came up with a bone structure that matched our actor's face and linked it to the markers, rather than using the markers themselves to deform the mesh (Figure 5). Thus, for each head, we just moved the bones to match the face, regardless of the polygon count. As a side benefit, we were able to assign to the markers varying amounts of influence over the bones. For example, some of the motion capture files had a little too much eyebrow motion. To fix this, I wrote expressions for the eyebrow bones that scaled back their movement to a percentage of their respective markers' motions. Here are some other problems that we faced:

EYES. Obviously, we couldn't place any markers on the face actor's eyelids or eyes. Nonetheless, we found that creating eye blinks was straightforward once we got the timing right. Later in development, we arrayed and moved blink keys to more natural positions.

The eyes themselves can be difficult to create at first. Try creating a half sphere, set the pivot point just a little in front of the center, then assign a "look at" (how appropriate) controller to the eyes. Another trick that's helpful is to create an invisible target object in the scene a few feet in front of the head, and force the character's eyes to track it. Once you take the creepiness out of your character's look, it's extremely easy to adjust where the character looks by simply moving the target object around.

JAW. The jaw is easily deformed by defining bones that are the length of the jaw bone. Their pivot points should be where a jaw's joint actually is (directly below the ears), with expressions or controllers that point them towards the chin.

MOUTH. We had problems moving the mouth; the face model would tear in between the bones. We solved the problem by creating longer bones and setting their pivot points close to the corners of the mouth. Next, we assigned a "look at" expression to the the next closest bone to the center of the lip. This structure assured a smooth continuity between the bones and prevented the geometry of the mouth from tearing — even in extreme circumstances.

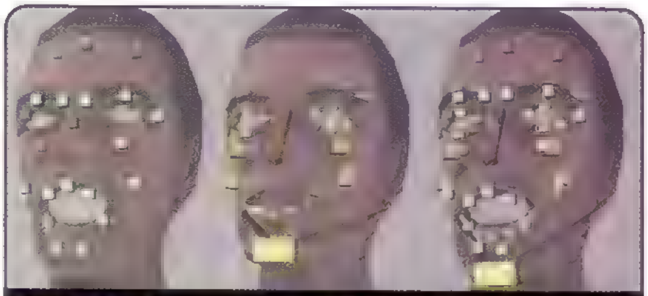


FIGURE 5. *The markers on the face, the bone structure at a neutral pose, and the combination of the bones and markers.*

It's Not in the Game Yet

We spent most of our time cleaning up the motion capture data and applying it to the geometry. At this stage, you'll find out how well you planned. The clean-up and application of data requires a few people (affectionately called the Tweak Department in Figure 3), who are typically animators. These individuals need a very high threshold for pain. Tweaking animation data isn't the most glorious job, but it is a very necessary production step that requires hard work.

If you've never worked with facial motion capture before, I suggest conducting the following test. First, go to a motion capture studio and spend a day capturing facial animations. Have your team clean up the data and apply it to your skeleton. See how well the data deforms the face, and then put it all into your game engine. Document how long each step takes and how much it costs. This practice will give you (and/or your boss) an idea of how the process works and what it involves.

This Is Not a Postmortem

It's difficult to figure out a reasonable solution for facial motion capture. After attending several SIGGRAPH talks earlier this year on the subject, I suspect that many left more confused than when they arrived. Furthermore, as you research the pros and cons of motion capture, almost invariably the bulk of your information comes from someone attempting to sell you some very expensive hardware or software. The hardware people will try to convince you that you need to spend big bucks on a new optical system, when all you need are some generic motions that you could buy from House of Moves. On the other hand, motion capture studios aren't always clear about their pricing schemes.

The bottom line is that regardless of the amazing hardware and software you use, everything rides on your actor's abilities. The acting is what the audience sees. You don't want to spend any time covering up bad acting — make sure the moves are right before the reflectors leave the face. Today, of course, the tools affect the outcome of a motion capture shoot, and I look forward to the day when the actors are in charge and the motion capture tools are an afterthought. ■

Lambsoft

Platinum Pictures

Kaydara

Adaptive Optics Associates

Motion Analysis

Vierte Art

Techimage

Sven Technology

DreamForge's SANITARIUM

by Chris Pasetto

SANITARIUM is an original adventure game filled with madness, delusion, and personal anguish. The same can be said for the development cycle. SANITARIUM was developed by DreamForge Intertainment Inc., set in the heart of Greensburg, Pennsylvania. DreamForge employs about 45 people working on three to four projects at a time. We had previously developed an adventure game entitled CHRONOMASTER, and our staff has a strong background in the development of computer RPGs. SANITARIUM was a landmark game for us, primarily because the project originated in-house, and we had so much of ourselves invested in it.

SANITARIUM

Chris Pasetto is the co-writer of SANITARIUM. He can be found at your neighborhood bar, wearing a kimono made out of a Dukes of Hazzard bedsheet and repeatedly screaming, "These are not my pants!" He can neither walk nor chew gum. He can be reached via e-mail at chrisp@dreamforge.com.

The Designers' Tale

SANITARIUM was born of a simple desire to create something special, something different. About two years ago, a few of us at DreamForge were feeling burnt out. We were tired of bandwagon games, eager for a fresh concept that we could dig into with enthusiasm. Most important, we wanted to make a fun game with substance and soul.

So, over some lukewarm cheeseburgers, Chris Straka (Director of Creative Development) asked Chad Freeman (Lead Programmer), Jason Johnson (3D Art Coordinator), Mike Nicholson (Lead Art and Design), Eric Rice (Art Director), and Tracy Smith (Post-Production Art Coordinator) what games they liked and why. Ideas were offered, counter-ideas brought forth, cheeseburgers grew cold and at times were nibbled upon. We discussed other forms of entertainment that would pertain to the game we wanted to make. Movies such as *Jacob's Ladder*, *Seven*, and *12 Monkeys* were mentioned repeatedly, television shows such as *The Outer Limits* and the original *Twilight Zone* episodes were discussed. At a certain point in the discussion, it became clear that we wanted to make an adventure game.

However, predictably, each of us had his own ideas of what the game should be about. Once the sound of human heads cracking together reached a deafening pitch, Chris Straka suggested that we make a game incorporating all of the ideas. He drew a crude wheel on a piece of paper, with a central hub and spokes radiating outward. The spokes would eventually become the diverse worlds within the game. The hub, that all-important plot framework that linked those worlds, had yet to be decided upon. Soon we realized that those separate ideas could be played out as psychotic episodes seen through the eyes of a mentally disturbed character. From that point on, the project was code-named Asylum. This would have been the game title, but we later discovered that the name was in use elsewhere. Hence the game was called SANITARIUM.

Once the design process started in earnest, we had SANITARIUM on the brain twenty-four hours a day. Each of us put a lot of fist-clenching, heart-soaring, spleen-churning effort into the project. It was a rewarding process for us, because most members of the team were new to the design experience. Sometimes at the end of the day, loose ends remained — questions regarding some story element or problems with the configuration of a certain puzzle. Many a wide-eyed game designer went to bed with visions of gargoyles and deformed children dancing around his head. When morning came, new angles and twists would reveal themselves like spirited flashers in the dawning sun.

We knew that we were on to something good. Though the core story was an afterthought in those original design meetings, we were determined to create a main plot line that held the game together and evoked strong emotions in the player. Working from disparate design notes, Mike Nicholson, the art and design lead, assumed the monumental task of scripting the game dialogue and creating the dialogue trees. As if he'd been suddenly transplanted into a Roger Corman movie, Mike quickly found himself neck deep in awkward lines and weak characters. After several days of confusion, he realized that the problem resided in the game worlds themselves. They had no true history, thus making it impossible to create detailed, realistic dialogues for the worlds' inhabitants. He went back to the design document and wrote background stories for each of the worlds, fleshing out the underlying themes and character motives, and smoothing over any inconsistencies. Mike also pushed the Sarah/Max connection and drafted the infamous "death scene." When he read his proposal to the design team, three of them nearly cried. With a concrete story in place, the characters all had rich backgrounds from which to draw and the same reference points to which they could refer. Scripting from that point on became relatively easy.

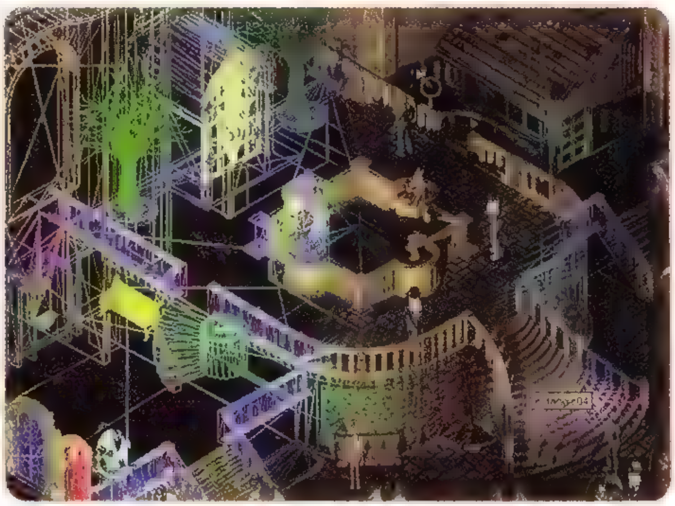
The original SANITARIUM design team: (left to right) Eric Rice, Jason Johnson, Mike Nicholson, Tracy Smith, Chad Freeman, Chris Straka, and Scot Noel. (The author is off to the side, chasing squirrels.)

After Mike put together a rough draft of the script, DreamForge hired Chris Pasetto as the project's writer. His primary responsibility was to refine all character and cinematic scripts. As the development process went on, the scripts became more complex (as we noticed things we'd missed) then simple (as we tried to streamline the dialogues). Eventually, the script files looked like a slaughterhouse tatters of butchered text casually strewn about like soggy meat by-products.

To maintain a consistent flow of game play and story, Chris had to balance the amount of dialogue that occurred

SANITARIUM

DreamForge Intertainment Inc.
Greensburg, Penn.
(724) 853-0200
http://www.dreamforge.com
Team Size: 37 men and women who no longer feel any pain
Release date: March 1998
Time in Development: 16 months
Critical tools: 3D Studio MAX, Adobe After Effects, Adobe Photoshop, Adobe Premiere, Cool Edit Pro, DeBabelizer Pro, FileMaker Pro, Inprise's Delphi, Inprise's Paradox, Microsoft Project, Microsoft Word, Smacker video codec, Sound Forge, Strata MediaPaint, Visual C++, Visual SourceSafe.
Target platforms: Windows 95



during any one non-player character interaction with how NPCs were distributed throughout the levels. During beta testing, testers complained that many of the dialogue interactions were too long and that the keyword-based interaction trees were sometimes too complex. In addition, characters could end up talking about subjects that seemed strangely out of order, jumping between disparate topics like a bad news segment. Since a lot of us here tend to work that way anyway, we didn't find it too confusing.

Travis Williams, our executive producer, insisted that we trim some of the encounters and link many keywords together to prevent confusion. A lot of the original dialogue was purely atmospheric and time-consuming for the player to wade through in search of real answers. The final version had an improved narrative flow and better pacing through a balance of dialogue and action.

We were constantly concerned that the emotional content of the game would be lost in the medium. Our goal was to give the player the creeps. We took the time to think out what we wanted the player to feel on each level, what message and mood we wanted to get across. Our efforts in creating a cohesive atmosphere included not just a good storyline, but an immersive audio experience as well.

Sanitarium was DreamForge's first product to utilize stereo sound. The point-sourced sound system made for very natural-sounding effects within each level. But technology alone couldn't make the sound great without the right people to take advantage of it. Steve Bennet, our music and sound effects composer for the project, did some awesome work with the soundtrack. Working from lists generated by the design team, Steve searched a huge sound CD library for the necessary effects. Then he processed the sounds using Cool Edit Pro and Sound Forge, sometimes working over a sound effect multiple times to match the atmosphere of the level. The moody music he added, entirely original compositions created on a Kurzweil keyboard, brought a definite style to the levels that enhanced the creepy atmosphere.

Nonetheless, the voice acting should have been better. It's difficult to compete with other developers who have access to name actors and meet everyone's expectations. This isn't an excuse, but a simple matter of economics. Would we have liked to have, say, James Earl Jones for the voice of Morgan? Of course. But with 80 NPCs and a limited budget for voice acting, big-name actors were an impossibility.

The final hurdle in the design process came from our publisher. Late in the project, during beta testing, ASC Games approached us with a significant design change. Dave Klein, the president of ASC Games, was wholeheartedly behind the project and loved our game. But… "Could you make it easier to play?" He explained that ASC Games wanted Sanitarium to have mass-market appeal and to be accessible to everyone, not just adventure game players.

Our faces turned Barney-purple with indignation. We felt that such a move would both compromise the game's sophistication and seriously jeopardize our completion of the project. We were a few weeks away from the final ship date and being asked to undergo a major revision of our basic approach to the game design. Also, we were stubborn.

Travis Williams came out to discuss what could be done and what couldn't be done in a reasonable amount of time. Our original approach to game play could be summed up as, "You're an adventure gamer. Figure it out." This new way of thinking forced us to ask hard questions, such as, "Where in the game is this information conveyed to the player?" In many cases, it simply wasn't. This led to a lot of easy fixes — having the main character utter a strategically placed bit of dialogue or even altering existing dialogue to help the player make puzzle connections. When this couldn't be done without a metric ton of contrivance, we adjusted the puzzles to be more user-friendly.

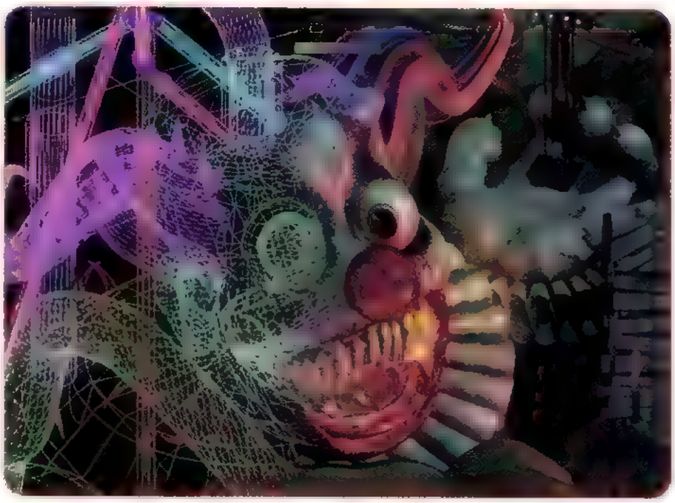
Admittedly, the changes made SANITARIUM focus more on entertainment than frustration. Players aren't perpetually stuck on difficult puzzles, so they participate in the story at a consistent pace and are able to enjoy it. Even the hardcore adventure game players that we initially targeted were satisfied by the balance of puzzle difficulty and richness of story.

The Artists' Tale

For all the designers' concentration on SANITARIUM's story, many of the game elements were conceived in artistic terms. We knew that the visual atmosphere of the game would be extremely important to the game play. The art conveyed the emotions that the player would feel, as well as the player character's state of mind. Because it's all about emotions and states of mind, SANITARIUM is a very art-intensive game. Thus, early in the process, the design team spent a lot of time determining the correct look for each part of the game.

One of the first things that we did was to gather reference material. We went on field trips to cemeteries, took pictures of St. Vincent's Cathedral, and raided local libraries. Eric Rice even captured a picture of a haunted gravestone on one of our cemetery photo shoots.

Back in the office, the heavy-duty work was getting underway. From concept sketches to full 3D models to touched-up game art, we strove to maintain that disturbing, realistic visual style as much as possible.



One of the first hurdles was an accurate isometric camera view. Finding a way to render six by four screen widths of landscape from twenty-four viewpoints and seam the shots together without any perspective warping was daunting. Tracy Smith worked out the bugs on this one. The final solution was to pull the cameras back to what would be the equivalent of viewing a city block with the Hubble telescope.

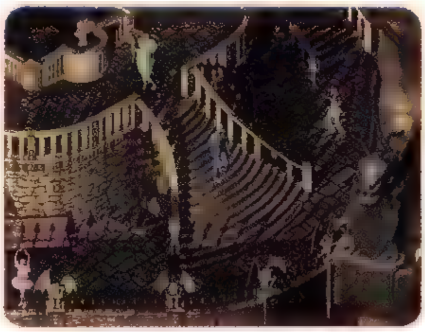
The biggest bottleneck that occurred during SANITARIUM's development came during the post-production of the art. We call the post-production department "5D," not because they exist on some H.P. Lovecraft pentadimensional plane, but because they work on a combination of 3D and 2D art. Once materials such as screens, characters, and animations poured smoothly out of 3D like good scotch, they had to go through the 5D twelve-step program before they would be ready for programming.

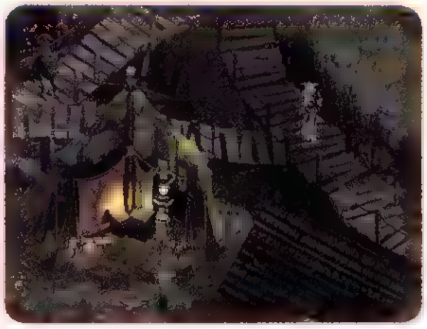
For the game art, Jason Johnson coordinated DreamForge's art staff as they used 3D Studio MAX to make the designers' vision a reality. The artists retouched the 3D background in Photoshop, then generated a temporary palette. Still barriers were clipped in true color, then squeezed into the temporary palette; coordinates were determined. 3D animations underwent alterations, retouches, and special effects as necessary. The artists then composited the animations into the retouched background. A final palette was generated and applied to all artwork for any given level. It was tough to create a palette that could support the massive environments and all the NPCs. The enormous number of colors used in the game was a nightmare for our post-production team. Using DeBabelizer Pro, these guys had to reduce entire levels of true-color renders to less than 230 colors. At that point, the original artists would walk over and ask, "Hey, what did you do to my level?" or management would say, "Is it gonna look like that when it's done?"

The steps continued. Still barriers in the temporary palette were reformatted into the new palette. Animations were then clipped and coordinates determined. Free-walking NPCs were retouched and clipped. Cursors, icons, and inventory were retouched and clipped. The player characters were put into a 24-color palette, retouched, and clipped. This was mind-numbing work at times. Even as brains turned to protein-rich pudding and limbs lost all feeling, the game art was taking shape.

All of this took anywhere from 50 to 350 man-hours per level. It was a demanding set of tasks requiring not only technical skill but the experience of having worked on games before and knowing how to deliver game art to a programming team in a perfectly usable form. Problems arose mainly due to inexperience.

The final look and quality of the levels and animations in SANITARIUM is a testament to some very determined artists who stayed late, worked weekends, and apologized when they were too sick to crawl to their desks





The Programmers' Tale

At first, we planned to convert an existing adventure game engine. Initial prototyping showed that this was about as likely as an Oscar nomination for Jackie Chan. So, we set out to create a new engine, re-using existing source code wherever possible. By using source code from earlier finished products, we avoided absolutely all bug-hunting hassles. Ha ha. That's a bald-faced lie. But using proven code did give us less to worry about — we knew it *had* worked at some point.

The basic engine was completed early in the project cycle, leaving us plenty of time during the rest of the project to sprawl in great big chaise lounges and sip tequila from fishbowls. Actually, that's not true either. Once the basic engine was complete, most of our time was spent on level building. A level scripting system based only on actions, flags, and flow control simplified the work. The rest of our time (those hours normally reserved for basic human functions such as sleep) was devoted to interaction scripting and special case programming, including blow-up puzzles and action areas. We had originally planned to create a blow-up puzzle editor, but time did not allow it. Reaching into the wiry guts of the beast, we hand-coded each blow-up puzzle instead.

Deciding on a codec for cut scenes was like clipping a lion's toenails. The first codec we looked at, True Motion, provided better visual quality but lacked support and ease of implementation. The second codec, Smacker, had just the opposite traits. We couldn't wait around for somebody to achieve the balance of properties that we needed because: first, we didn't have time; and second, we didn't know if anyone would get it right any time soon. The final decision came late in the development process: support and ease of implementation won out over visual quality.

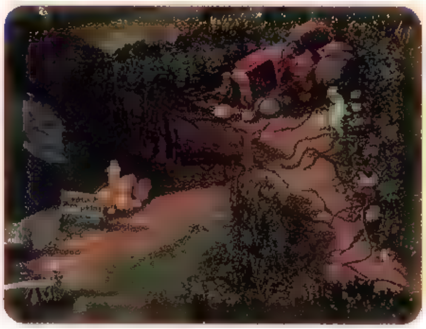
For SANITARIUM's bug testing, we entirely divorced ourselves from tracking bug reports on paper. Both our in-house testers and the test team at ASC Games used FileMaker Pro 4.0 to generate, sort, and track the status of all bugs. Because we were both using the same software, we were able to trade databases with ease, do proper triage, compare priorities, and eliminate duplication. SANITARIUM was DreamForge's most thoroughly tested product to date.

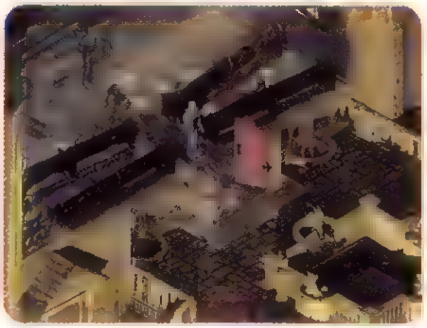
But even with this extensive testing regimen, the game still shipped with the infamous "lockout bug" on Level 2. If the player wandered around the town long enough and fulfilled certain conditions, it became impossible to enter any of the buildings. This was a big disappointment. In the countless man-hours of testing between ASC Games and DreamForge, no one encountered this bug. Herein lies a valuable lesson, grasshopper. There is simply no test group more likely to find a crash bug than those tens of thousands of initial buyers.

The Sweet

SANITARIUM represented a significant success for DreamForge in several areas. Many of these have to do with our personal sense of accomplishment in making this game, but others are things we learned along the way.

1. **BRING IT ON HOME.** As a game that was designed in-house, an enormous amount of energy and personal pride went into SANITARIUM. Remember that this project began as a few guys hanging out after work saying, "Wouldn't it be cool if we made the game that we would enjoy playing?" Even after months of work, we weren't sure if the game would ever reach the shelves. As the team's hard labor began





to bear fruit, the whole company's energy and enthusiasm grew, sustaining us through the crunch periods. That sense of personal ownership in a product cannot be underestimated. It defined the experience of SANITARIUM for us as developers.

Not only that, but our staff offered us an inexpensive focus group. Fairly early on in the project, the ideas we designers had been bouncing off one another seemed like stale old superballs. We needed more outside opinions to give the project perspective. We invited small clusters of DreamForge employees to join us in the design room. Like a nightmare ride at Disney World, we took these groups through the game puzzle by puzzle, plot twist by plot twist. Questions and comments gave us valuable information — what looked interesting and what seemed confusing.

As a result of those walkthroughs, it became painfully clear that Level 6 of our design just wasn't cutting it. It was as though Al Gore had walked into the room. People's eyes glazed over at that point in the walkthrough; yawns were abundant. We looked at each other and said, "This is not good." So we asked people, What would be clever, interesting, and creepy? Bugs seemed to be the answer. Based on staff input, the resulting Level 6 of SANITARIUM is much stronger and enjoyable than our original design.

2. HOME MOVIES. From the very beginning of the development cycle, we wanted to give SANITARIUM a dark, cinematic feel. In most games, the cut scenes are treated like a necessary evil or worse — a pageant of plug-ins *du jour* meant to dazzle viewers and draw their attention away from the game play. We were determined to establish a style for the cinematic cut scenes, to make them an integral part of the game. We especially wanted the flashback cut scenes to deliver an emotional impact to the viewer, because they dealt directly with Max's life, love, and suffering. To support that idea, we shot the scenes to mimic the letterbox look of movies. Our cinematic coordinator, Marty Stoltz, drew upon his filmmaking background to guide us in precise cinematic screen direction. Joe Skivolocke also lent his post-production expertise to the effects for all memory cinematics. A lot of work went into ensuring correct camera usage and post-production of cinematic scenes — especially for the flashback sequences, which were meant really to touch the player emotionally.

All cinematics came into the world as storyboards — carefully laid out in Adobe Premiere and passed on to the artists. The 3D staff worked from storyboards to create raw .AVIs. Our post-production team worked with these .AVIs, touching up the rough edges and applying special effects using Adobe Photoshop, Adobe After Effects, and Strata MediaPaint. Some of the raw .AVI material had to be thrown out in the end (because it didn't work, because something looked wrong, because one of the lighting crew members was eating a sandwich in the background). Still, our shooting ratio was about 3:1 (for every second of cinematic material that we used, 3 more seconds were tossed out) — that's pretty good when you consider that the average movie has a 20:1 shooting ratio. The polished cut scenes that went into the game were the best DreamForge had ever produced, anchored thematically by a unique and consistent vision.

3. MODULAR FURNITURE. We had all worked on other games and were familiar with the potential threat of cutting levels, puzzles, and whatever else seemed expendable when the crunch was on and no amount of Mountain Dew could keep us going. From the beginning of the design, we prepared for such eventualities by

structuring the game in a modular fashion. We constructed the game in portions that would add to game play and advance the story, but wouldn't detract from the game overall if they were taken out. In the end, we were able to keep the amputations to a minimum. A big combat zone and some blow-up puzzles took a trip through the plumbing, but otherwise the cuts were fairly minor. Modular design at the start of the project ensured that the final game would remain true to its initial vision.

4. Hey, nice assets. As an experiment, lead programmer Chad Freeman implemented an asset management system utilizing a Paradox database, which centralized all of the game assets in one place. Tools developed in Delphi and Visual C++ accessed the assets from this database. This solution provided several benefits. For one thing, we could easily analyze the asset data and take appropriate actions when total asset size broke the budget for a level. We could also view filenames and descriptions of individual assets. The database system let us group assets by levels or by other criteria. A single game level had hundreds of art and sound files. Searching for a particular asset by the filename alone would have been the equivalent of finding the fat guy wearing the Star Trek shirt at a sci-fi convention. Naturally, the ability to sort through assets quickly saved time and energy.

Because this process was experimental, we weren't able to fully exploit the database system. For example, the programmers were the only ones who utilized the system during the level-creation process. However, Chad Freeman would eventually expand the system so that artists and sound technicians could add assets to the database and level creators could access them directly from there, eliminating redundant file storage. In addition, the system could also store level information, allowing these same types of reporting, sorting, and other benefits to be extended to the levels themselves. Overall, the development of Sanitarium never made full use of these database management tools. As game content grows larger and larger (DVD and beyond), using database tools for data storage will help developers more and more.

We also utilized Visual SourceSafe for the first time during Sanitarium's development. Historically, programmers have been beset with the extremely time-consuming and tedious job of hand-merging code. Never again. Like a divine beam of light shining into our otherwise dank and shadowy cubicles, SourceSafe made code merging far easier and more reliable. SourceSafe also has other benefits, including the ability to keep a precise revision history of your code, so that you can painlessly retreat from the inevitable "bad move" programming-wise.

We chose SourceSafe specifically because it allowed multiple check-outs; the structure of our C files prior to adopting SourceSafe was such that it was common for more than one person to be working on a single file at the same time. SourceSafe also allows a project to be branched off, letting one person work on a demo while another continues development of the game. The projects can later be re-merged, so that fixes in the demo can be integrated into the main source.

5. Project management for the insane. Using Microsoft Project and his own devious tracking tables prepared in Microsoft Word, project manager Scot Noel would recalculate our progress every week to two weeks. This method accounted for the progress of every single game element, from blow-up puzzles to art fixes to code implementations. GANTT charts demonstrated the flow of work between departments and individuals. These enabled us to respond promptly to the most critical problems by showing how the late delivery of a particular asset might throw off the final ship date by days or weeks.

Critical paths were plotted using PERT charts in Microsoft Project. Upon seeing these Daedalean webs of near-infinite complexity, many of us felt that Scot had gone, finally and irrevocably, insane. But once we penetrated the mysteries of the PERT chart, we saw the value of tracking the sensitive interdependencies of tasks through critical paths. As different departments, or even particular individuals, caught up with one another or moved ahead of expected schedules, the critical path would change. Armed with this knowledge, Scot could walk up to any given programmer and say, "The critical path for this game is going right through you at the moment."

Such monitoring helped direct the pressure and motivate the right people, letting others go home and get a good night's sleep. As are all systems, the PERT charts were imperfect. Some people always seemed to be on the critical path, most notably Chad Freeman, the game's lead programmer. All of us here at DreamForge hope that Chad will be able to leave the hospital soon. We already have a respirator set up alongside his desk.

6. Publisher buy-in. Our publisher, ASC Games, believed in what we were trying to accomplish and provided valuable input throughout development. They behaved as if they were buying into our vision rather than just purchasing it. For the most part, they took a hands-off approach, and only required changes that they were convinced would significantly improve the quality and salability of the game. Travis Williams, SANITARIUM's executive producer, put a lot of heart into the project — not to mention all the cool prerelease games and toys he sent us. We were so grateful, we put his head in the game.

We were very pleased with ASC Games' strong commitment to marketing SANITARIUM. The box is a work of art in itself, and the rule book has received praise for its strength and simplicity. The magazine ads are impressive and true to the spirit of our game. One simple act for which the team is eternally grateful: ASC Games' marketing department didn't give away the game plot on the box or in the manual. It's always a relief when you don't see the central mystery of your game printed in big red letters across the back of the game box.

The Sour

While SANITARIUM represents a phenomenal success for us here at DreamForge (both professionally and personally), there were some unfortunate stumbling blocks along the way. We keep telling ourselves: that which did not kill us has made us stronger. Never mind the scar tissue.

1. Animations. Due to the size of the game, each character had a limited number of animation frames. In many cases, this caused the movement to look stiff and unnatural. Looking back on it, we would have preferred smoother animations with more angles — especially for the main character, Max. If we had taken this into account earlier in the project, we might have had an opportunity to fix it. By the time we realized that eight angles looked a little stiff, it was too late. The limited angles also caused problems for players trying to navigate Max through the levels. He'd often get stuck on corners, then either walk in place like some demented mime or frustrate the player with a litany of, "Can't go that way."

Getting consistent lighting between the characters' standard animations (status quo, walk, use, and so on) and the specific animations requiring interaction with the environment (such as kicking in the school door) was another nightmare. Different artists did these animations months apart, and this was a constant battle from beginning to end. A huge amount of time was spent fixing things as opposed to advancing the project.

2. Combat zones. The action sequences needed more attention. They were important for guiding the pace of the story, but didn't have the feel that we were after in the end product. The original idea behind these areas was based on one of DreamForge's earlier titles, VEIL OF DARKNESS. It had wonderful combat areas that helped break up the pacing between the puzzles. However, in SANITARIUM, multiple factors forced us to water down the combat zones or in some cases cut them altogether. We had originally planned a large combat zone for the Hive level of the game. We'd hoped to make "Grimwall vs. the Hive" one of the most fun and integral combat areas, but it was cut from the game for various reasons.

3. Sturm und Drang. When you have a company of forty to fifty people, it's impossible to do anything without rubbing someone the wrong way. SANITARIUM was put together by a design team that in part came together naturally and in part was hand-picked by Chris Straka, our head of creative development. Individual employees (usually Chris) designed our previous titles. But we decided that team design would be the way to go. While team design has the potential to fracture the unified vision of a game, the various team members ultimately complement each other's interests and goals, lending more depth to the game. This is a nice way of saying, "The team argued a lot, but came up with better solutions as a group."

Difficulties didn't end there. Once SANITARIUM became a full-time project, many staff members complained, "Why didn't I get a chance to be on the design team?" Quite a bit of friction was generated because people felt as though they'd been snubbed. Unfortunately, a design team reaches critical mass once it has more than six members. Design teams work in much the same way as clown cars. Too many people cramming themselves into the

design would have certainly brought an already volatile process to a grinding halt. At the same time, SANITARIUM'S personnel power structure created inequalities between staff members that were never meant to happen.

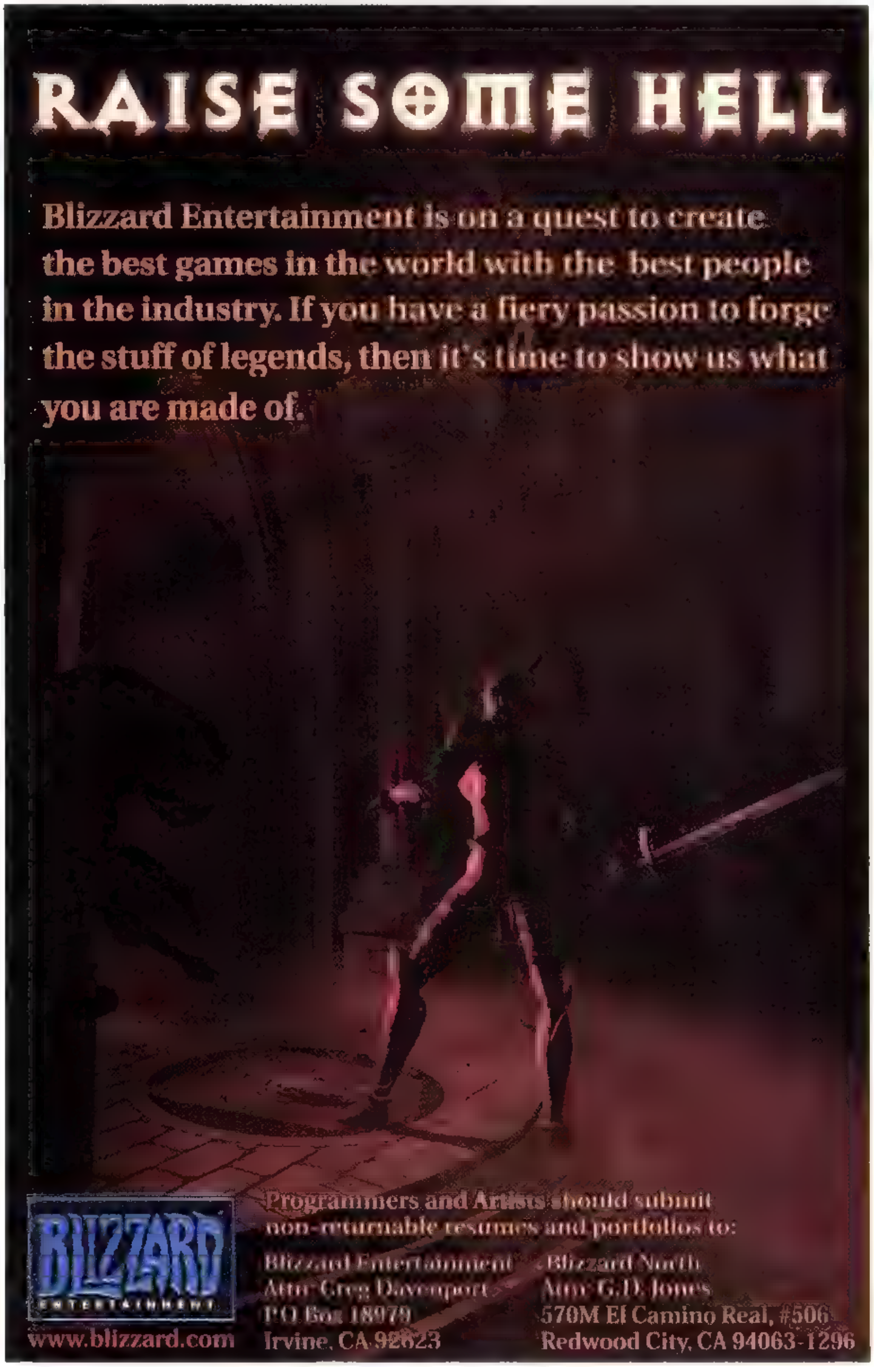
DreamForge learned much about design teams from SANITARIUM, and we're having great success with the design team setup in our current projects. We've taken steps to recognize people's desire and initiative, and have parceled out responsibilities to those individuals willing to take up leadership positions. Now, rather than saying, "Why wasn't I included?" everyone moans, "Why did I get into this?" It's great fun.

4. LOAD TIMES. While long level loading times were an accepted design limitation from the beginning of the project, a system that could better manage memory and allow for streaming of more data from the CD could have benefited the game. The tight schedule left such a system impossible to pursue. A more sophisticated memory-management scheme could have allowed for shorter initial loading times, larger levels, and so on.

5. NEW KIDS IN THE CUBE. SANITARIUM is a huge game. A lot of people worked on it, meaning additional efforts had to be taken to coordinate and organize everyone's labor. Familiarizing people with the vision of the game from an artistic and design point of view was a real challenge. Sometimes, keeping everyone on the same page seemed to be a chore, especially as new people came onto the project.

A lot of time was spent getting people to understand the status of the project and the direction in which it was headed. A new artist would ask, "Why am I making this one-eyed guy?" and we'd say, "Didn't anyone tell you?" We made the mistake of projecting time schedules as if new hires, following a brief training period, would be as competent as our most experienced people. Some of those experienced people were performing administrative and training tasks, and thus weren't producing much art. Art delivered by our new people often had problems when it went to the programmers. This meant doing things twice, sometimes three times. Projections and flow charts slid downhill, taking into account the flow of asset delivery, identification of problems, correction of problems, and re-implementation of assets. Since art delays were slowing down programming, we tried to use temporary art. This didn't work out because creating useful temporary art for the programmers proved to be nearly as time-consuming as the real thing. And just to throw a little cherry on top of the three-layered cake of delays, we lost two artists during production.

Even though SANITARIUM had a longer production time than any previous DreamForge project, there were still some things that we would have liked to tweak or add to make it better. As it was, we went through a lot of crunch periods in order to get things done on time. The sheer amount of artwork required for the game nearly overwhelmed us. Unfortunately, due to the nature of the game, delays in artwork had a snowball effect because the level implementers needed the actual artwork in order to set up their levels. ■


RAISE SOME HELL

Blizzard Entertainment is on a quest to create the best games in the world with the best people in the industry. If you have a fiery passion to forge the stuff of legends, then it's time to show us what you are made of.

Programmers and Artists should submit non-returnable resumes and portfolios to:

Blizzard Entertainment
Attn: Greg Davenport
P.O. Box 18979
Irvine, CA 92623

Blizzard North
Attn: G.D. Jones
570M El Camino Real, #506
Redwood City, CA 94063-1296

www.blizzard.com

STAINLESS STEEL STUDIOS

Join The Team That's Developing Premier Real-Time Strategy Games For The Next Millennium

Stainless Steel Studios, Inc., founded by Rick Goodman (creator/lead designer for Microsoft's Age of Empires®), and based in Cambridge, Massachusetts has embarked on the development of real-time, multi-player strategy games for the next millennium.

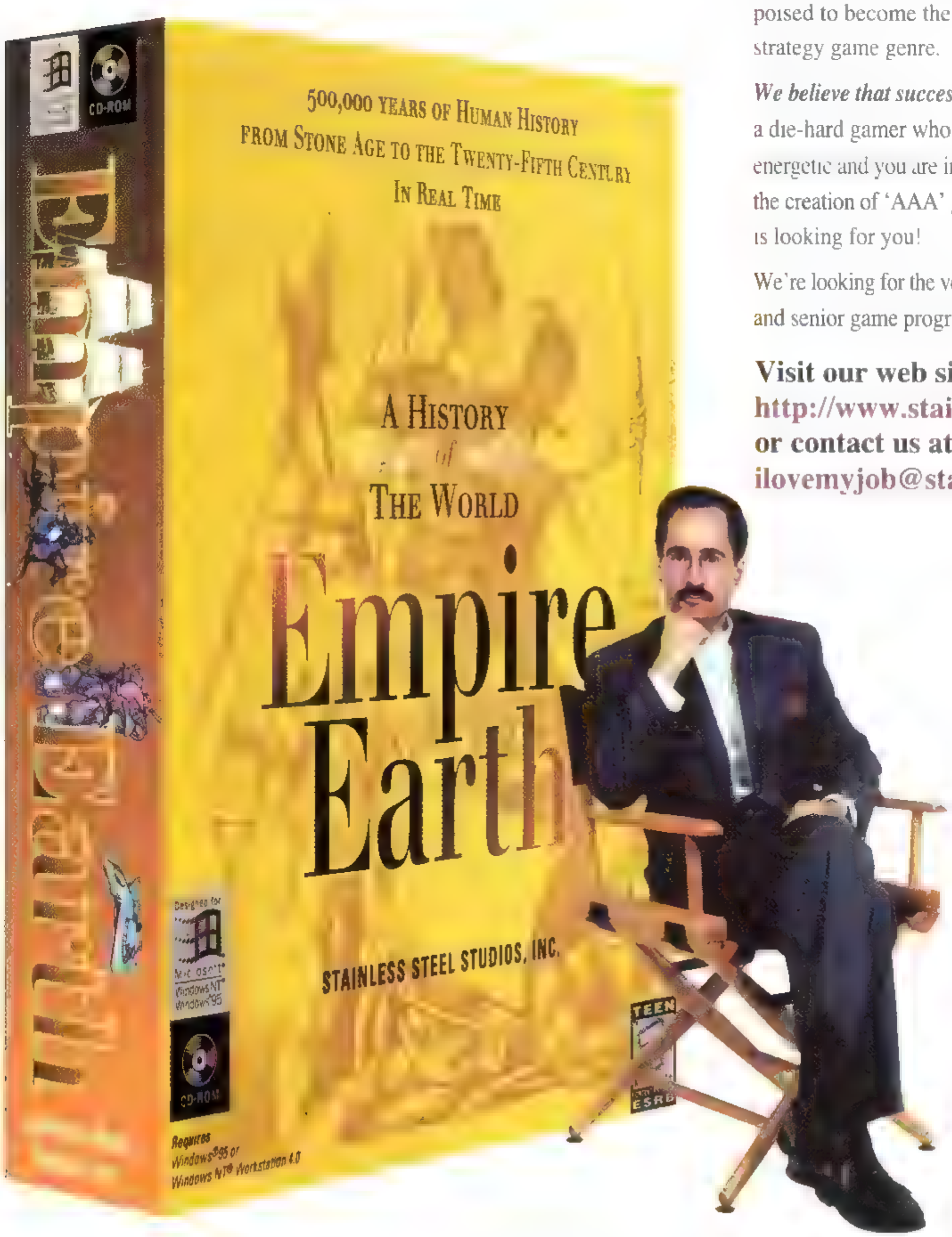
Here's a unique opportunity to participate in the creation of innovative, top-quality game titles and become part of a new, fast-growing game development company that is poised to become the industry leader in the real-time strategy game genre.

We believe that success is based on teamwork. If you are a die-hard gamer who is hard working, self-motivated, energetic and you are interested in having major input into the creation of 'AAA' game titles, Stainless Steel Studios is looking for you!

We're looking for the very best artists, character animators and senior game programmers in the industry.

Visit our web site:
http://www.stainlesssteelstudios.com
or contact us at:
ilovemyjob@stainlesssteelstudios.com.

All inquiries will be kept strictly confidential. We are an equal opportunity employer offering an excellent salary and benefits package


500,000 Years of Human History
From Stone Age to the Twenty-Fifth Century
In Real Time
A History of The World
Empire Earth
Stainless Steel Studios, Inc.
CD-ROM
Designed for Microsoft® Windows NT® Windows®95
Requires Windows®95 or Windows NT® Workstation 4.0
TEEN
ESRB
Empire Earth

Rick Goodman:
Recipient of the 1998 Computer Game Developer Conference's "Annual Achievement Award for Game Design and Development".

Microsoft and Age of Empires are either registered trademarks or trademarks of Microsoft Corporation. ©1997 Microsoft Corporation. All rights reserved

TREMOR™
ENTERTAINMENT

WANT TO WORK WITH THE BEST?

SO DO WE!

CAN YOU PROGRAM A TOP TITLE?

WOULD YOU LIKE A SHARE OF THE PROFIT?

WANT TO SET YOUR HOURS?

WELL? WHAT ARE YOU WAITING FOR?

818-729-0020
BURBANK, CALIFORNIA
(Junior Programmers need not apply.)
RWATTERS@TREMOR.NET

JOIN US

SENIOR 3D GAME PROGRAMMERS
SENIOR 3D MODELERS/ANIMATORS
3D ARTISTS/LEVEL DESIGNERS

THE COLLECTIVE IS LOOKING FOR TALENTED, EXPERIENCED, AND FOCUSED INDIVIDUALS WHO ARE ADEPT IN THE BLACK ART OF GAMES DEVELOPMENT AND DREAM OF WORKING IN A COOL, CHALLENGING, FUN, AND ZERO BS ENVIRONMENT.

THE COLLECTIVE DEVELOPS LEADING EDGE TITLES FOR THE INDUSTRY'S TOP PUBLISHERS. CURRENT PROJECTS INCLUDE STAR TREK: DS9™ AND AEON FLUX™ FOR GT INTERACTIVE.

PLEASE SUBMIT YOUR RESUME AND SAMPLE CODE, DEMO REELS OR LEVEL LAYOUTS TO:

THE COLLECTIVE
2600 E. PACIFIC COAST HWY, SUITE 260, CORONA DEL MAR, CA 92625
FAX (949) 717-2513 EMAIL: JOBS@COLLECTIVESTUDIOS.COM
WWW.COLLECTIVESTUDIOS.COM

Star Trek: Deep Space Nine: TM & © 1996 Paramount Pictures. All Rights Reserved.
STAR TREK and related marks are Trademarks of Paramount Pictures.
Aeon Flux TM & © 1996 MTV Networks. All Rights Reserved.

Digital Anvil

FORGE ON WITH US!

VISIT OUR WEB SITE AT:
www.Digitalanvil.com

Digital Anvil is a multimedia entertainment software company born out of the talents of Chris Roberts, creator of the top selling Wing Commander series, a top creative team from the software and film industries and the resources of Microsoft. Digital Anvil was formed to create innovative games and movies with high impact storylines and mass audience appeal. We are always looking for creative men and women who share our vision of the potential interactive entertainment.

Artist

Programmers

Designers

Send resumes to: **careers@digitalanvil.com** or fax to the attention of Evan Brandt (512) 457 - 0404 **www.digitalanvil.com/low/jobs**

- Are you a "Serious Gamer"?
- Are you looking to create the hottest games in the market?
- Are you looking to become part of entertainment history?

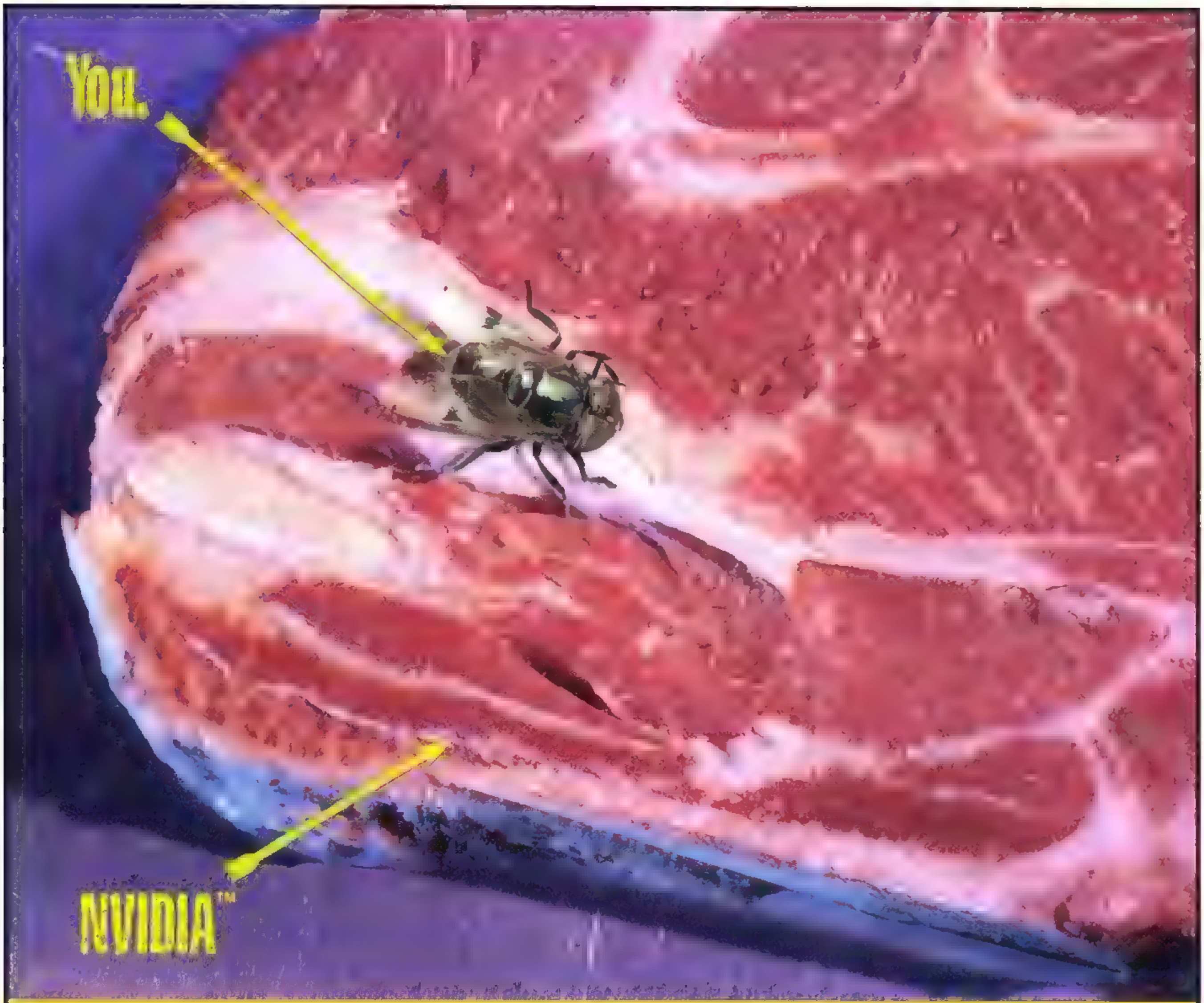
SINGLETRAC

SingleTrac, a GT Interactive studio developer of the hit video games "Twisted Metal™", "Critical Depth™", and "Jet Moto™", is looking for talented people to join our highly skilled development teams.

Sound Software Engineer
Sound Designer
Network Assistant
Expert Modeler*
Expert Animator*
Software Engineer

Competitive salary and benefits
Send resumes and/or demos to:
SingleTrac
Attn: Human Resources
175 West 200 South
Salt Lake City, UT 84101
Fax: (801) 521-5547, Email:

*All Animator and Modeler positions require a demo and/or portfolio.



If you're looking for a world-class 3D experience, it's dinner time.

Feast your eyes on the world's leading 3D processor company, NVIDIA. The company that served up the award-winning RIVA family is now looking for a few hungry people to bring its latest processor, RIVA TNT,™ to the table. Boasting over 7 million transistors, the RIVA TNT will deliver a stunning visual experience to the mainstream PC market. If you're into serious 3D, NVIDIA is the place to be. Welcome.

Hardware

- Circuit Design Engineer
- Logic Design Engineer
- Product Test Engineer

Software

- 3D Device Driver Engineer
- Windows/Window NT Display Driver Engineer
- Q/A Test Engineer

Marketing/Sales

- Product Marketing
- Technical Product Marketing
- International Sales

We offer an intense and fun environment, great benefits and competitive compensation. Apply online at www.nvidia.com; email your resume to hr@nvidia.com; fax to (408) 615-2800; or mail to: Professional Staffing, NVIDIA Corporation, 3535 Monroe Street, Santa Clara, CA 95051. EEO/AA.

NVIDIA
Bringing 3D to the world™

www.nvidia.com

Top 3D Programming & Art Talent
PSX, N64, DreamCast & PC Computer Games

Are you respected for your creative vision and talent and interested in joining a development team with a ten year history of award-winning titles? Stormfront Studios is looking for individuals who'd enjoy working on first-tier games with top game publishers. Our titles include NASCAR '98, Madden Football, Star Trek: Deep Space Nine, Andretti Racing, Byzantine, La Russa Baseball, etc.

We're adding to our staff and looking for talented people for the following positions:

Racing Team: Technical Director, Lead PSX Programmer, Lead N64 Programmer, Lead PC Programmer, Sr. N64 Programmer, Entry N64 Programmer, Entry PSX Programmer. Baseball Team: Lead PC Programmer, Sr or Lead Internet Programmer, Lead Console Programmer, Entry PC Programmer. Role Playing Team: Lead PC Programmer. Game Show Team: Sr. PSX Programmer. Action Team: 3D Max Artist with 2D Talent.

Ours is an open environment of small, mixed-discipline development teams where high standards are respected and your contributions will be recognized. We offer a very competitive compensation and benefits package.

Please send your resume to:

Stormfront Studios, Attn: M. Daglow, 4040 Civic Center Drive
San Rafael, CA 94903 Fax: 415-461-3865, e-mail: mdaglow@aol.com
www.stormfrontstudios.com

AIMING TO REACH 30,000 PROFESSIONAL GAME DEVELOPERS?

TARGET PAVILION HITS THE SPOT!

Game DEVELOPER

CALL
Ayrien Houchin at
415.905.2788
to advertise

COIN-OP IS A REAL RUSH!

- Cutting edge technology
- Defining new game experiences
- A cashbox full of coins tells the whole story!

Opportunities for programmers, animators and others seeking real arcade action.

Send resumes and demos to:
resumes@agames.com
fax: (408) 434-5888

Atari Games Corporation
Human Resources
675 Sycamore Drive
Milpitas, CA 95035

www.atarigames.com

ATARI

Specializing in music & sound design for the gaming industry

Planet Z Music

Scoring for 3d animations

In-game music

Interactive music

Large sfx library/original sound design

Editing, mastering, and batch processing audio files/format conversion

Major title experience

978.348.4789

www.planetzmusic.com

DIGITAL MEDIA CENTRE @ Seneca

3D

- 16 Week Full-Time 3D Digital Animation Training
- Softimage/Maya/Houdini
- Motion Capture
- Starts February & June 1999

Seneca College of Applied Arts & Technology
Tel: (416) 491-5050 ext 4445
Toronto, Canada
http://dmc.senecac.on.ca dmc@senecac.on.ca

•*OSAP loan eligible*

Model by student: Michael Langford

Puppet Works

Silicon Graphics Computer Systems

Technical Animations, Inc.

888-44-PIXEL

"Any order over $1300, FREE Meshpaint"

3D MAX R2.5	Call	Lightwave 5.6	1690
AnimationMaster	160	Lightwave Educ	830
CharactrStudio2	Call	Form-Z / RZ	Call
After Effects-Win	430	AfterEffectsProd	1330
TotalTrain'g[AE]	550	FinalFxComplet	650
UltimatteNT/Mac	1260	Boris EffectsPro	430
DiscreetL Paint	Call	DiscreetL Effect	Call
Speed Razor 4s	1690	SpeedRazorRT	Call
SpeedRazorUpg	520	Inscriber Prods.	Call
Digital Fusion 2	Call	Lightscape NT	390
Adobe Products	Call	Chyrn Concerto	1890
Inscriber Prods.	Call	Artbeats	Call
Elastic Reality 3	940	MorphMagic	450
Davinci 3D Max	550	RayMax 2	290
DirtyReyes	380	JetaReyes Max	300
MetaReyes Max	530	CartoonReyes	380
ClothReyesMax	530	Reyes 5-Pack	1440
Clay Studio Pro	350	BonesPro Max	440
LumeTools [LW]	720	TrU-V [Intel]	280
Aura [Newtek]	610	Aura Educationl	390
Perception RT	Call	MatroxDigisuite	Call
DPS Perceptn	1570	CanopusDVRex	Call
DPS Capture	790	DC50 PinlSysts	1670
MicroScribe 3D	2470	Oxygen 402	1590
STB-Symmetric	Call	Oxygen 202	840
GlydrTXGold16	390	ElsaGloria X/XL	Call
L3100 Leadtek	Call	Accel GMX2000	Call

INTERNATIONAL ORDERS WELCOME

Call For Full Price List with Latest Pricing
Tel: 847-297-1000 / Fax: 847-297-4820
Website>> www.techanim.com
Email>> sales@techanim.com
[VISA - MC - AMEX - NOVUS - COD...]

SD '98 COM BOOTCAMP

Seminars in Building COM/DCOM Components with Visual C++, MFC and ATL.

5 Days of Intensive Hands-On Training

Seattle	Oct 5–9
San Jose	Oct 12–16
Arlington, VA	Nov 9–13
Boston	Nov 16–20
Atlanta	Nov 30–Dec 4

Contact us today! www.sdexpo.com/1998/bootcamp
800-441-8826 • 415-905-2702 • bootcamp@mfi.com

Enlist in a five day, hands-on SD '98 Bootcamp and learn how to build powerful and flexible programs with COM and DCOM.

EXPERT INSTRUCTOR:

Richard Hale Shaw

TEAM DISCOUNT!

Register 5 more people at the full price and the 6th person attends FREE! All registrants also receive a free SD collectors t-shirt! Pricing available at our website or call for a brochure!

SD DEVELOPMENT

Miller Freeman

Create your own cool 3-D Games!

3D GameStudio

Authoring System
for 3-D Realtime Computer Games

- ◆ Create cool 3-D adventures **without programming!**
- ◆ Contains World Editor and the **ACKNEX** 3-D engine
- ◆ Free 3-D template game with 150+ textures included
- ◆ 3-D polygonal landscapes with slopes, bridges, buildings
- ◆ Player can walk, drive, fly, climb, swim, dive...
- ◆ Self-defined panels, overlays, cockpits, and scrolling text
- ◆ Create your own objects, actors, walls, weapons, menues
- ◆ 320x400 256 color smooth scrolling VGA resolution
- ◆ 8-channel stereo sound & midi support
- ◆ Imports PCX, LBM, MDL, WAV, MID and IBK files
- ◆ 200+ pages English manual with game tutorial

No Royalties!!

3D GameStudio Lite $99
3D GameStudio Commercial $199
(+SVGA+2-Player mode)
3D GameStudio Professional ... $1450
(+Polygonal Actors+CD-Audio+FLIC player)

Prices +$20 for overseas shipping / $40 for additional email delivery
Infos, demos, ordering → http://www.conitec.com

CONITEC Germany ■ D-64807 Dieburg ■ Dieselstr. 11c
Tel +49 6071 92520 Fax 925233 ■ www.conitec.com

Editorial Resources

Product	Company	URL	Phone	Page
3D Studio MAX	Kinetix	http://www.ktx.com/	(415) 547-2000	25, 53, 57
Acid	Sonic Foundry	http://www.sonic foundry.com	(800) 57 SONIC	12
After Effects	Adobe	http://www.adobe.com	(408) 536-6000	57
Cool Edit Pro	Syntrillium	http://www.syntrillium.com	(602) 941-4327	57
Character Studio	Kinetix	http://www.ktx.com/	(415) 547-2000	53
DeBabelizer Pro	equilibrium	http://www.equilibrium.com	(415) 332-4343	57
Delphi	Inprise	http://www.inprise.com/delphi	(408) 431-1000	57
FileMaker Pro	FileMaker	http://www.filemaker.com	(800) 986-2249	57
FiLMBOX	Kaydara	http://www.kaydara.com	(514) 842-8446	52
Ghost	Symantec	http://www.symantec.com/sabu/ghost/index.html	(408) 253-9600	72
Glide	3Dfx Interactive Inc.	http://www.3dfx.com	(408) 935-4400	6
ImageBlaster	Altiris	http://www.keylabs.com/software/rdeploy/index.htm	(801) 377-5484	72
Maya	Alias|Wavefront	http://www.alias.com	(800) 447-2542	53
MediaPaint	Strata	http://www.strata.com/html/mediapaint.html	(435) 628-5218	57
Microsoft Project	Microsoft	http://www.microsoft.com	(425) 882-8080	57
Microsoft Word	Microsoft	http://www.microsoft.com	(425) 882-8080	57
MorphMagic	Platinum Pictures	http://www.platinumpictures.com	(608) 355-1097	52
Paradox	Inprise	http://www.inprise.com/paradox	(408) 431-1000	57
Photoshop	Adobe	http://www.adobe.com	(408) 536-6000	57
Polytrans	Okino Computer Graphics	http://www.okino.com	(905) 672-9328	6
PowerAnimator	Alias|Wavefront	http://www.alias.com	(800) 447-2542	53
Premiere	Adobe	http://www.adobe.com	(408) 536-6000	57
ProMotion	Lambsoft	http://www.lambsoft.com/smirk.html	(612) 813-3737	52
RTIME Interactive Networking Engine	RTIME	http://www.rtimeinc.com	(206) 281-7990	6
Smacker	RAD Game Tools	http://www.radgametools.com/smkmain.htm	(425) 893-4300	57
Smirk	Lambsoft	http://www.lambsoft.com/promotion.html	(612) 813-3737	52
Softimage	Softimage	http://www.softimage.com/	(514) 845-1636	25
Sound Forge	Sonic Foundry	http://www.sonic foundry.com	(800) 57 SONIC	57
SurfaceSuite	Sven Technologies	http://www.sven-tech.com	(650) 852-9242	53
Visual C++	Microsoft	http://www.microsoft.com	(425) 882-8080	57
Visual SourceSafe	Microsoft	http://www.microsoft.com	(425) 882-8080	57
VTune	Intel	http://developer.intel.com/design/perftool/vtcd	(800) 253-3696	10

ADVERTISER INDEX

NAME	PAGE	NAME	PAGE
3Name 3D	33	Numerical Design	14
Atari Games Corp.	68	Nvidia	67
ATI Technologies	19	Okino Computer Graphics	33
Aureal Semiconductor	3	Planet Z Music	69
Blizzard	64	QSound Labs Inc.	27
Cakewalk	23	RAD Game Tools Inc.	C4
The Collective	66	Raleigh Group International	5
Conitec Datensysteme Gmbh.	70	Savannah College of Art and Design	71
Digital Anvil	66	Seneca College	69
The Duck Corp.	21	Softimage	C2
GT Interactive	66	Stainless Steel Studios	65
IBM	7	Stormfront Studios	68
Immersion Corp.	9	Technical Animation	69
Macrovision	29	Tremor Entertainment	66
Metrowerks Inc.	C3		

Continued from page 72.

working with images. This scenario is represented in Table 3.

By comparing the figures in Table 3, you can see that the cost difference between rebuilds and images amounts to a total savings of $306.68 for just one machine in one single week. If all machines could be tested with eight operating systems, the savings would be $3,066.80 in labor alone for one week. Using the CD-ROM method with existing CD-ROM mastering equipment, just two workstations nearly pay for the setup expense in a week. Not to mention the savings gained by not purchasing duplicate hardware that will become obsolete in another month

These are elementary figures based on sequential work. When a system is rebuilt manually, the employee must sit in front of the machine to be available to answer installation questions. With images, employees can start a rebuild and move to the next machine, and the next, and the next. Also consider the flexibility of building test beds for each scenario when testing different drivers for the same video card. The possibilities are endless. Put your own scenario in here and see if using the IQATBS method wouldn't save you time and money in your labs. ■

TABLE 1 [illegible]

Task	Unit Cost	Total Cost
9GB+ hard disk space on the LAN	$1,000 each	$1,000
Network cards for each system	$50 each	$500
Imaging software license per machine	$30	$300
Boot floppies	$1	$10
Labor to setup and image ten systems	$40	$400
Total Setup Cost		**$2,210**

TABLE 2 [illegible]

Task	Unit Cost	Total Cost
Network cards for each system	$50 each	$500
Imaging software license per machine	$30	$300
Boot floppies	$1	$10
Labor to setup and image ten systems	$40	$400
CD-ROM Media	$5	$50
Total Setup Cost		**$1,260**

TABLE 3 [illegible]

OS Tested	Time for rebuilds	Cost of rebuilds	Time for images	Cost of images
Windows 95 Gold	4:00	$40	0:10	$1.67
Windows 95 SR2	4:00	$40	0:10	$1.67
Windows 95 Plus	4:00	$40	0:10	$1.67
Windows 95 SR2 Plus	4:00	$40	0:10	$1.67
Windows 98 Gold	4:00	$40	0:10	$1.67
Windows 98 Plus	4:00	$40	0:10	$1.67
Windows NT 4	4:00	$40	0:10	$1.67
Windows NT 5 BETA	4:00	$40	0:10	$1.67
System Total	**32:00**	**$320**	**1:20**	**$13.32**

COMPUTER ART
Savannah College of Art and Design

SOAPBOX

by Jeanne Collins

Turbo-Charge your Test Lab

Let's face it, Microsoft may have a corner on the operating system business, but that doesn't mean we have any fewer operating systems or software configurations to support.

NT Workstation may cohabitate with either Windows 95 or Windows 98, but have you tried getting Windows 95 and Windows 98 to live together? Don't bother trying — it's not going to happen.

So, how can a meager game developer have access to lots of operating systems and multiple test beds on a limited budget? The answer is: images! There are several products now available that will give you the ability to make a sector-by-sector image of a hard drive and/or hard drive partition. Here are the two with which I am most familiar:

- ImageBlaster. A module in Rapid Deploy by Keylabs' new spin-off software company, Altiris (Preferred Product mention in the 1997 FLA Awards). http://www.keylabs.com/software/rdeploy/index.htm.
- Ghost. (Which we recently purchased). By Symantec. http://www.symantec.com/sabu/ghost/index.html.

Pricing for these products is based on a sliding scale that takes into account their usage, so it's best to contact the publisher's sales team for pricing. These manufacturers gear their products more towards the hardware builder who builds several computers with the exact same components. However, these products are a dream-come-true for quality assurance test labs who need to maximize their budget dollars in order to test a wide variety of operating systems and installation scenarios.

Let's take the typical game development test lab as an example. If they're lucky, they have 15 to 20 test machines. (Hey, I said "lucky.") They've done their best to amass the widest variety of hardware components possible within these 15 to 20 machines. But with such a relatively small number of machines, it's impossible to have every potential configuration.

In terms of operating systems, a test lab needs Windows 95, Windows 98, and Windows NT 4.0. But rebuilding each system every single time those operating systems need testing consumes hours and hours of labor time, and completely obliterates productivity. (The preceeding list of operating systems doesn't even take into account the imminent release Windows NT 5.0, and that all games currently in testing should be tested on the version of that system as well.)

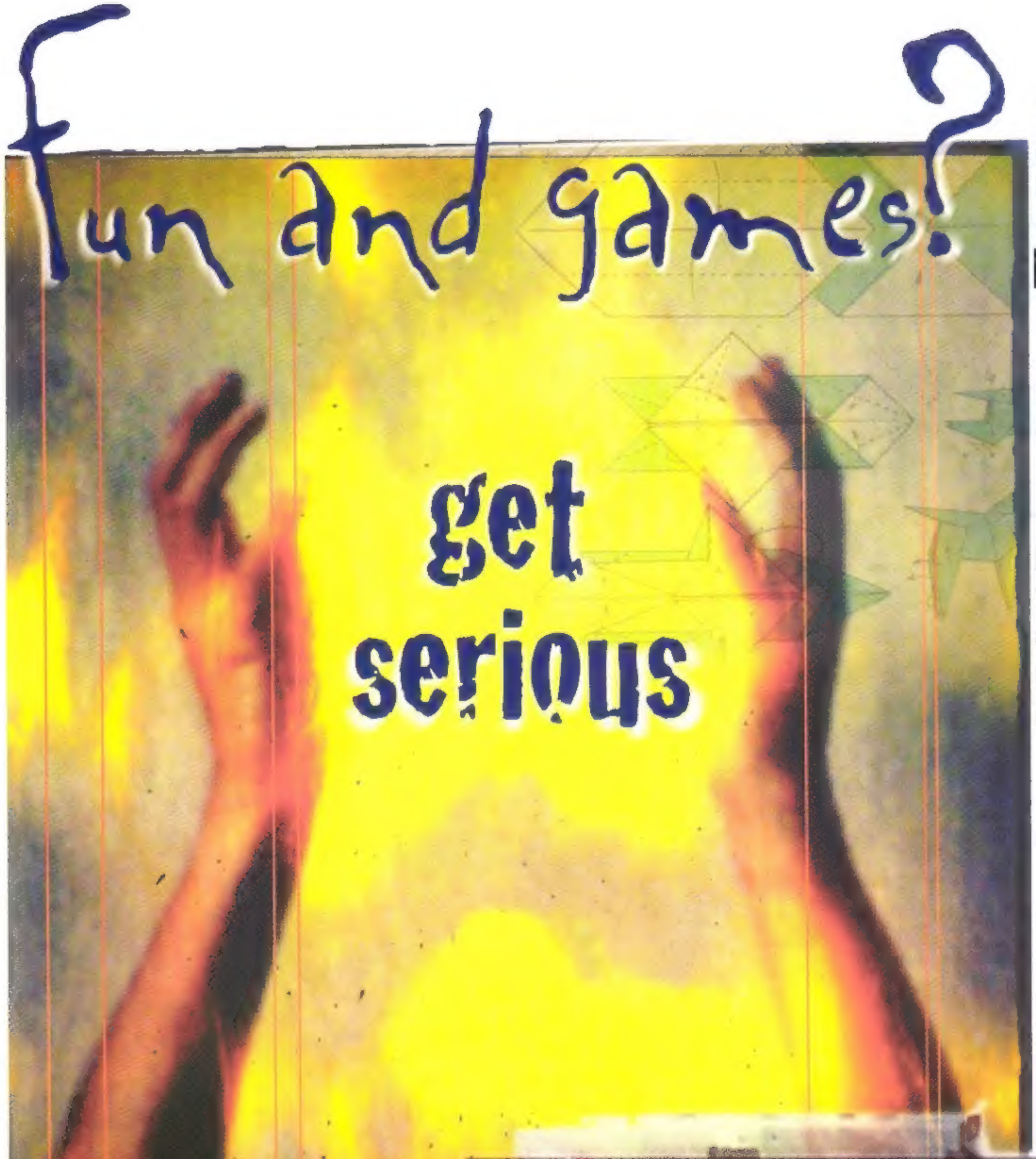
The biggest obstacle to adopting an Imaging QA Test Bed System (IQATBS) that I've heard is "We can't afford it." My answer to that is "You can't afford not to!" Take a look at Tables 1 and 2 (see page 71).

With these estimates, there is one image for each machine, and room to add several more images for each machine at the cost of approximately $45-$49 per image (including labor, image licenses, and materials). Now, where are the savings?

Compare those costs to the testing time lost by constantly rebuilding systems to recreate usable test beds without stored images, or the costs of buying enough hardware to have a separate machine for each scenario. And don't forget that those systems would need to be rebuilt every time the test bed is altered by the installation of a new build. Let's look at a testing scenario as an example. It takes place in the same ten-computer lab, and involves testing one station with eight operating systems over the course of one week. We'll compare the costs and productivity of working with rebuilds vs.

Continued on page 71.

Jeanne Collins is a quality assurance manager at GTE Internetworking. She is sometimes referred to as a "self-proclaimed evangelist for quality assurance in the gaming industry." Send her feedback at jeanne@im.gte.com.



You've got a game in production and you're building the tools for the next one. **This is no time for playin' around.**

Get serious with industrial-strength tools from the Factory — CodeWarrior Professional and CodeWarrior for Playstation.

Here's why more than 100,000 programmers are dead serious about CodeWarrior:

- Cross targets: game consoles, PC, and Macintosh
- Wickedly fast build times
- Easy-to-use GUI interface
- State-of-the-art graphical debugger
- Support for AMD-3D™ technology

CodeWarrior®

www.metrowerks.com
1-800-377-5416
info@metrowerks.com

metrowerks®

seriously

Game images courtesy of Ion Storm.

The Best in Game Development Technology!

Smacker Video Technology

Smacker
VIDEO TECHNOLOGY
3
NEW
3.2 VERSION!
QuickTime and MMX
(up to 400% faster) Support
Also contains several new
blitting styles.

Smacker is a compressor for video, animation and sound data designed specifically for games. Smacker has been used in all aspects of game design: cinematics, cut-scenes, video-sprites, transparent videos, image decompression, scrolling video backgrounds, and more. It has been used in over 1,100 games because it is fast (real fast!), easy-to-integrate, and available for most game platforms.

The Smacker SDK is available for DOS, 16-bit Windows, Windows 95, Windows NT, Win32, Mac, and PowerMac. The Smacker SDK API is identical across the platforms and includes everything necessary to playback videos with synchronized sound.

For graphics under Windows, Smacker supports CreateDIBSection, WinG, DispDIB, and DirectDraw. For the Mac, Smacker supports both GWorlds (with lightning-fast assembly blitters to augment CopyBits) and direct-to-screen decompression. For DOS, Smacker has built-in support for VESA 1.x (direct decompression into banked video RAM) and VESA 2.0 (linear frame buffer support). Since Smacker can decompress into any linear piece of 8-bit or 16-bit memory, using it with your own graphics code or a third party library (such as MGL) is no problem.

Smacker also includes optimized assembly memory-blitters for 1x, 2x, 2x interlaced, and 2x smoothed (interpolated) modes to both 8-bit and 16-bit color surfaces (system or VRAM). Included are MMX and non-MMX versions for PC, and 68K and PowerPC versions for Mac.

For sound, Smacker has built-in support for the Miles Sound System for DOS and Windows, DirectSound for Windows 95 and NT, the Windows waveOut system, HMI's SOS library for DOS, Diamondware's STK, and Sound Manager for the Macintosh.

The Miles Sound System

MILES
SOUND SYSTEM
NEW
5.0 VERSION!
MPEG Layer-3 Decompression!
New 3D sound support -
uses DS3D, RSX, or A3D!

New MPEG Layer-3 decompression support!
Miles 5.0 includes a redistributable MPEG Layer-3 decoder (patent rights from Thomson and Fraunhofer). MPEG Layer-3 provides perceptually lossless compression at 11 to 1! Your sound files will be more than **ten times** smaller and will sound exactly the same! Miles supports both on-the-fly decompression (for minimal memory use), or pre-playback decompression (for minimal CPU hit). Miles 5.0 even supports MP3 compression of DLS instrument data for incredibly small MIDI files.

New digital system! Faster mixing, filtering, & plug-in support!
Miles 5.0 includes a completely new digital audio subsystem. It includes a faster mixer (with MMX support), on-the-fly filtering, and a new plug-in system for adding run-time sound effect processors which can be called at the pre- or post-mix stage.

New high-level 3D sound support!
Miles 5.0 has an all new high-level 3D sound API which supports DirectSound3D, Intel's RSX, and A3D (selectable at run-time). Using the Miles 5.0 3D API, you can have 3D audio in your game in as few as three calls! More importantly, you won't be tied to a specific low-level 3D API, so you can use the technology that makes your game sound best.

Of course, Miles 5.0 provides all of the features you loved in previous versions: digital mixing, interactive MIDI with DLS support, hard drive/CD-ROM streaming, red book CD-audio, powerful callbacks, full DirectX support, and much more!

Smacker and the Miles Sound System are licensed on a per-product or per-site basis with NO ROYALTIES WHATSOEVER!

Partial list of Smacker and/or Miles Sound System customers:

Activision, Blizzard, Eidos, Ion Storm, Microsoft, Electronic Arts, Bullfrog, Lucas Arts, Access, Bungie, DreamWorks, Humongous, Kesmai, Monolith, Origin, Psygnosis, Papyrus, Accolade, 3DO, Acclaim, GT Interactive, Hasbro, Maxis, Sierra, Westwood, Sony, Take 2, The Learning Company, New World Computing, The Neverhood, Core Design, Blue Byte, Dynamix, Microprose, Zombie, Viacom, Hypnotix, Boffo, Impressions, Animatek, Stormfront Studios, Virgin Interactive, Krisalis, Sir-Tech, Atari, Byron Preiss, Time Warner, Software Sorcery, McGraw Hill, Looking Glass, Capstone, Adrenalin, Spectrum Holobyte, IntraCorp, Strategic Simulations, Software 2000, Bethesda, Compton's New Media, Firestorm, Criterion, Interactive Magic, SCi Limited, Trilobyte, Adeline, Rocket Science, Epic, Legend, MECC, Velocity, Capcom, Sculptured Software, Software Toolworks, Sega, and Knowledge Adventure.

... over 2,300 games - see our web site for title lists!

Call for discounts when you license both Smacker and the Miles Sound System together.

425-893-4300

335 Park Place Suite G109
Kirkland, WA 98033
425-893-4300
FAX 425-893-9111
sales@radgametools.com
www.radgametools.com

©
RAD GAME TOOLS

Made with love by

RETROMAGS

Our goal is to preserve classic video game magazines so that they are not lost permanently.

People interested in helping out in any capacity, please visit us at retromags.com.

No profit is made from these scans, nor do we offer anything available from the publishers themselves.

If you come across anyone selling releases from this site, please do not support them and do let us know.

Thank you!